KfK 5234
August 1993

An Approach to the Machine Front-End Services for the CIM-Open System Architecture (CIM-OSA)

Wu-Nan Hou
Institut für Angewandte Informatik

Kernforschungszentrum Karlsruhe

KERNFORSCHUNGSZENTRUM KARLSRUHE

Institut für Angewandte Informatik

KfK 5234

An Approach to the Machine Front-End Services for the CIM-Open System Architecture (CIM-OSA) *)

Wu-Nan Hou

*) von der Fakultät für Informatik der Universität Karlsruhe genehmigte Dissertation

Kernforschungszentrum Karlsruhe GmbH, Karlsruhe

Kernforschungszentrum Karlsruhe GmbH
Postfach 3640, 76021 Karlsruhe

ISSN 0303-4003

ABSTRACT

This thesis presents an approach to the development of ***Machine Front-End Services*** within the framework of ***CIM Open System Architecture (CIM-OSA)***.

CIM-OSA is in development by ESPRIT AMICE projects since 1986. It defines an integrated methodology to support all phases of a CIM system life-cycle from requirements specification, through system design, implementation, operation and maintenance. CIM-OSA provides a ***Modelling Framework*** and an ***Integrating Infrastructure (IIS)***. The Modelling Framework supports the modelling of business activities of an enterprise. The Integrating Infrastructure is an operating infrastructure supporting the execution of CIM-OSA models. With both Modelling Framework and Integrating Infrastructure, CIM-OSA enables a consistent and complete information processing from the process design to the manufacturing.

The Integrating Infrastructure comprises of four blocks of services: the *Communication Services* for the management of the liaison with communication subsystems and transparency mechanism; the *Business Services* for the control of execution of CIM-OSA models; the *Information Services* for the system-wide information exchange and the *Front-End Services* for the integration of enterprise resources. In CIM-OSA, three types of enterprise resources are distinguished: machine control programs, human interactive programs, and application programs. These resources are reflected in the ***Front-End Services***, comprising of three elements, namely ***Machine, Human and Application Front-End Services (MF, HF and AF)***.

The dissertation was initiated during the validation of CIM-OSA, the main objective of the ESPRIT project VOICE. It was found that there are no products nor specifications available for the Machine Front-End. The Machine Front-End should, on one side, take the control and management of execution of CIM-OSA elementary Function Models, and on the other side, integrate heterogeneous manufacturing devices. In the IIS environment of client-server model, the Machine Front-End has a two-fold function: as a client and as a server.

The proposed approach provides a ***Control Model Library*** and a ***Control Engine***. The *Control Model Library* contains the application-specific control knowledge of enterprise functions (called *Functional Operations* in CIM-OSA Terminology), while the generic control mechanism is the kernel of the *Control Engine*. The separation of

these two basic elements is crucial for the MF development in such an environment with multiple clients and servers.

The approach makes use of the concept of *Service Units* which may invoke services of the international standard MMS (*Manufacturing Message Specification*) or the proprietary services. It provides an easy way for the migration of existing proprietary applications into a CIM system and also leads to the fulfilment of user requirements. Furthermore, it applies the *object modelling technique* to specify the MF capability, and uses the principle of the complementary interaction model to define the interaction between the Machine Front-End and its Clients.

Based on these ideas, the Machine Front-End has been specified in detail and implemented by use of a highly portable programming language, the C-language. A testing environment was established for the validation of the MF Prototype. It is not only used for this work to validate the proposed approach and to investigate the behaviour of the Machine Front-End in a CIM-OSA system, but also offers a good basis to implement the whole Integrating Infrastructure. The MF prototype has become an important part of deliverables for the ESPRIT AMICE project 5288.

The CIM-OSA concept was enhanced through the implementation of the Machine Front-End in the McCIM system. This work has made a considerable contribution to the evolution and credibility of the CIM-OSA concept.

This thesis begins with the introduction of CIM-Architectures in Chapter 1. It discusses various CIM models and the research activities in the CIM area, but the focus is on CIM-OSA. Chapter 2 describes the problem statement and the goals of this work. The concepts proposed for the MF design are given in Chapter 3 to 6, and the implementation and the results of testing are presented in Chapter 7. Some perspectives on further development of a CIM-OSA system are discussed in Chapter 8 and the important results of this work are concluded in the last chapter.

The text and figures in this thesis use a number of abbreviations and acronyms which are listed prior to Chapter 1.

Ein Ansatz für das Maschinen-Frontend der offenen CIM-Systemarchitektur CIM-OSA

Zusammenfassung

Diese Arbeit behandelt Entwurf, Prototypentwicklung und Evaluation eines Maschinen-Frontends in der offenen CIM-Systemarchitektur (CIM-Open System Architecture - CIM-OSA). Das Maschinen-Frontend dient der Integration heterogener Fertigungseinrichtungen.

CIM-OSA wird vom ESPRIT-AMICE-Konsortium schon seit 1986 entwickelt und definiert eine integrierte Methodik, die alle Phasen zum Aufbau eines CIM-Systems von der Anforderungsspezifikation über den Systementwurf, die Implementation, den Betrieb, bis hin zur Instandhaltung unterstützt. CIM-OSA umfaßt zwei Hauptkomponenten: ein ***Modellierungsgerüst*** und eine ***integrierende Infrastruktur***. Das Modellierungsgerüst dient der Beschreibung der Geschäftsprozesse eines Unternehmens, und die integrierende Infrastruktur der Ausführung der erstellten Prozeßmodelle. Mit Hilfe dieser beiden Komponenten ermöglicht CIM-OSA eine durchgängige Informationsverarbeitung vom Prozeßentwurf bis zur Fertigung.

Die integrierende Infrastruktur umfaßt vier Dienstblöcke: die Kommunikations-Dienste zum unternehmensweiten Datenaustausch, die Business-Dienste zur Interpretation der CIM-OSA-Prozeßmodelle, die Informations-Dienste zur Integration heterogener Informationssysteme und die Frontend-Dienste zur Integration der Unternehmensressourcen. In CIM-OSA werden die Unternehmensressourcen in drei Klassen untergliedert: Maschinensteuerungsprogramme, interaktive Benutzer-Ein/Ausgabe und Applikationen. Diese Ressourcen spiegeln sich in den ***Frontend-Diensten*** wider, die drei entsprechende Elemente enthalten, nämlich die ***Applikations-***, ***Human-*** und ***Maschinen-Frontend-Dienste*** (***MF***, ***HF*** und ***AF***).

Diese Arbeit wurde im Rahmen des ESPRIT-VOICE-Projektes begonnen, das die CIM-OSA-Konzepte in industriellen Umgebungen validieren sollte. Es wurde erkannt, daß CIM-OSA nur eine globale konzeptionelle Beschreibung des Frontends vorgegeben hat. Eine brauchbare Spezifikation lag nicht vor, und es existierte auch kein verfügbarer Lösungsansatz, welcher die Aufgaben des Maschinen-Frontends

realisieren konnte. Das Maschinen-Frontend muß in der Lage sein, einerseits die CIM-OSA-Grundfunktionen abzuarbeiten und andererseits die heterogenen Maschinensysteme zu integrieren. Außerdem hat das Maschinen-Frontend innerhalb der Client-Server-Architektur der integrierenden Infrastruktur eine doppelte Rolle, es ist sowohl Client als auch Server.

Der in dieser Arbeit vorgeschlagene Lösungsansatz unterscheidet sich von der üblichen Methode zum Entwurf von Client-Server-Anwendungen in der Trennung des applikationsspezifischen Steuerungswissens von den allgemeinen Steuerungsmechanismen. Dieser Ansatz erfüllt nicht nur die CIM-OSA-Rahmenbedingungen, sondern erleichtert auch die Implementation der CIM-OSA-Grundfunktionen.

Nach diesem grundlegenden Ansatz wird das Maschinen-Frontend in zwei Komponenten gegliedert: eine *Control Model Library* und eine *Control Engine*. Die *Control Model Library* enthält das spezifische Steuerungswissen der CIM-OSA-Grundfunktionen, und die *Control Engine* die allgemeinen Steuerungsmechanismen. Darüberhinaus wurde das Konzept der *Service Units* eingeführt, die mit den internationalen Standard-MMS-Diensten (Manufacturing Message Specification) oder mit herstellerspezifischen Diensten realisiert werden können. Ferner wurde die *Object Modelling Technique* zur Spezifikation des Dienstumfangs des Maschinen-Frontends verwendet und die Interaktion zwischen dem Maschinen-Frontend und dessen Klienten durch ein Paar komplementärer Interaktionsmodelle realisiert.

Basierend auf diesem verfeinerten Entwurf wurde das Maschinen-Frontend spezifiziert und in einer prototypischen Implementation realisiert. Die Anwendbarkeit des Konzepts wurde anhand des Prototypen mit mehreren Prozeßmodellen auf einem Testszenario erprobt. Es wurde erstmals der wichtige Aspekt von CIM-OSA, die "ausführbaren Modelle", demonstriert. Das Konzept sowie der erstellte Prototyp wurden vom ESPRIT-AMICE-Konsortium sehr gut angenommen.

Das CIM-OSA-Konzept wurde durch die Implementation des Maschinen-Frontends im McCIM-System aufgewertet. Diese Arbeit hat damit einen wesentlichen Beitrag zur Akzeptanz und zur Weiterentwicklung des CIM-OSA-Konzepts geleistet.

ACKNOWLEDGEMENTS

This work was carried out in the Institute for Applied Informatics (IAI) in the Nuclear Research Center of Karlsruhe (KfK). I would like to thank the KfK for the scholarship support during this research work.

I am grateful to Prof. Dr.-Ing. H. Trauboth and Prof. Dr.-Ing. U. Rembold for their useful advice and encouragement, and also Prof. Dr.-Ing. R. Dillmann for his evidence.

I am deeply indebted to Dr. E. Holler for his support and helpful advice to achieve this work, and to my colleagues Mrs. M. Didic, Dr. W. Molisz, F. Neuscheler, F.-J. Kaiser, M. Huber, L. Bogdanowicz, P. Gymer, and Ph. Guittot for many fruitful discussions and their help on the establishment of the test environment.

Also I greatly appreciate the engagement of Mr. M. Klittich from the Daimler Benz AG in applying this work in the ESPRIT AMICE Project.

Espcially, I am grateful to my family for their continued care and encouragement throughout the period of this research.

CONTENTS

FIGURES

List of Abbreviations and Acronyms

CIM-OSA Terminology:

AC	Activity Control (an IIS component)
AF	Application Front-End Services (an IIS component)
AFE	Application Functional Entity (a type of CIM-OSA conformed CIM-Module)
AFO	Application Functional Operation (a type of FO)
BC	Business Process Control (an IIS component)
BP	Business Process Model (Intermediate level of a CIM-OSA model)
CM	Communication Management (an IIS component)
DM	Data Management Services (an IIS component)
DP	Domain Process Model (Top level of a CIM-OSA model)
EA	Enterprise Activity (Bottom level of a CIM-OSA model)
FO	Functional Operation (elementary CIM-OSA Function Model). There are three types of FOs distinguished, namely: *Human*, *Application* and *Machine Functional Operation* (*HFO*, *AFO*, *MFO*)
HF	Human Front-End Services (an IIS component)
HFE	Human Functional Entity (a type of CIM-OSA conformed CIM-Module)
HFO	Human Functional Operation (a type of FO)
IEE	Integrated Enterprise Engineering (CIM-OSA build-time environment)
IEO	Integrated Enterprise Operation (CIM-OSA run-time environment)
IIS	Integrating Infrastructure (Operating platform of CIM-OSA system)
MF	Machine Front-End Services (an IIS component)
MFE	Machine Functional Entity (a type of CIM-OSA conformed CIM-Module)
MFO	Machine Functional Operation (a type of FO)
PR	Procedural Rule
RM	Resource Management (an IIS component)
SD	System-wide Data Services (an IIS component)
SE	System-wide Exchange (an IIS component)
B-Services	Business Services (an IIS main component)
C-Services	Communication Services (an IIS main component)
F-Services	Front-End Services (an IIS main component)
I-Services	Information Services (an IIS main component)

Additional:

AMICE reverse acronym for European Computer Integrated Manufacturing Architecture (an ESPRIT consortium)

API Application Program Interface

APU Application Program Unit (a type of Program Unit)

CIM Computer Integrated Manufacturing

CIM-OSA CIM-Open System Architecture (Result of the ESPRIT AMICE Projects)

ESPRIT European Strategic Programme for Research and Development in Information Technology

ExecCtrlFlag Execution Control Flag (an attribute of the ModelExecCtrl data object used for the control of model execution)

FTAM File Transfer and Access Management (an ISO standard protocol)

HPU Human Program Unit (a type of Program Unit)

IAI Institut für Angewante Informatik (Institute for Applied Informatics-KfK)

ISO International Standard Organization

KfK Kernforschungszentrum Karlsruhe GmbH (a research center in Germany)

MAP Manufacturing Automation Protocol, International Standard

MMS Manufacturing Message Specification (an ISO standard protocol)

ModelExecCtrl Model Execution Control Data Object (a data structure used for the control of execution of a model)

ModelExecCtrlList Model Execution Control List (a list of Model Execution Control Data Objects)

MPU Machine Program Unit (a type of Program Unit)

OSI Open System Interconnection (ISO Reference Model)

PDU Protocol Data Unit

PU Program Unit, a basic program function implemented in external program which co-operates with the Front-End Services to achieve a specified FO. According to the FO types three types of PUs are distinguished, namely: *Human*, *Application* and *Machine Program Unit* (*HPU*, *APU*, *MPU*)

RPC Remote Procedural Call (an ISO standard protocol)

SQL System Query Language

SU Service Unit

VDB Variables Description Block (for the description of the model variables)

VMD Virtual Manufacturing Device

VOICE Validating Open System Architecture in Industrial CIM Environment (an ESPRIT consortium)

1) CIM-Architectures

This chapter introduces various CIM-Architectures which have recently been published. It identifies requirements of the present manufacturing enterprises in building a CIM system and derives the capabilities which a good CIM-Architecture should have. Several CIM models defined by well-known computer companies are discussed. However, the focus of this work is on the CIM-Open System Architecture.

1.1) Introduction to Computer Integrated Manufacturing (CIM)

With the proliferation of the information technology, Computer Integrated Manufacturing (CIM) has been recognized as being a very important key for the success of a manufacturing enterprise. Among other factors such as labour costs, technology, and qualification/education of employees, CIM is one of the most promising opportunities for enterprises which intend to stay competitive and keep their position in the rapidly changing marketplace [*Pans90a*, *KaCE92*].

CIM can be defined as a computer-based information processing system which integrates all types of computer systems within a manufacturing enterprise. A CIM system covers all the business activities of an enterprise which are the decision support, production planning, automatic control of manufacturing processes, quality control, maintenance, stores, accounting, cost analysis, etc. A CIM system provides an appropriate integration of enterprise operations throughout the enterprise by means of an efficient information exchange. It aims at the reduction of development time and production costs, the reaching of a higher product quality and a better planning, and the usage of the available resources.

When building a CIM system the following requirements have to be met, thereby reducing some of the limitations of the present manufacturing enterprises:

- integration of 'islands of automation': Automation has been realized with locally optimized manufacturing systems in the last decades. The physical and logical connection of these systems is difficult because of different data formats used by the various business functions. This has led to a problem known as 'islands of automation' [*Stra89*].
- interoperability/portability of vendor-dependent devices/software: Nowadays, the equipment for manufacturing systems is supplied by a multitude of vendors. For

example, there are approxiamtely 200 manufacturers of industrial robots alone, each having a controller of its own choice of computer hardware [*ChDa89*]. Most of these devices have their own local programs and proprietary operating systems. Thus software represents a large percentage of the total automation cost, also maintenance is critical.

- support in overall optimization of enterprise operations: Most manufacturing systems are implemented to achieve specific functions accordingly to a bottom-up approach. A complementary top-down approach for the integration of all the enterprise operations is needed in order to optimize global manufacturing objectives [*Remb90b*].
- flexibility to respond quickly to changes in the enterprise environment: The continuously changing market and technology has enforced enterprises to keep their manufacturing processes in a new perspective of flexibility, adaptibility and reliability, instead of stability. The need for rapid adaptation of the manufacturing processes to the new enterprise environment becomes an important aspect.

Therefore, a good CIM-Architecture should be able to overcome these drawbacks and provide the following capabilities:

- support of enterprise modelling: This concerns the definition and manipulation of models used to describe the real world of manufacturing enterprises. The user should be supported with the design and implementation of his own CIM system.
- system openness: This concerns the international standardization efforts for hardware as well as for software application protocols, e.g. models, interfaces, communication protocols.
- (re-)use of standard software modules: This implies the application of standard software modules which can easily be adapted to a CIM system. The availability of such software modules will enable rapid adaptation of manufacturing processes to a new enterprise environment.
- support for the operation of a CIM system: This involves the facility of dynamic system configuration. An established CIM system should be reconfigurable in a changing enterprise operation environment.
- integration support: This entails the integration of existing 'islands of automation' and other proprietary applications into a CIM system. The existing investment can be thereby protected.

A lot of work has been done so far to achieve the goals listed above. Well known vendors as well as researchers contributed to this task. Actually, several CIM architectures were conceived by companies like IBM, DEC and Siemens.

Various CIM models have been discussed in [*Remb90a*], which focuses on the standardization efforts in CIM. The following sections briefly describe the basic structures and concepts of CIM architectures introduced by world-leading computer companies. There are also several CIM achievements from the research institutions. An example of such an achievement is the ***CIM-BIOSYS Platform*** (*CIM-Building Integrated Open Systems*) [*West90-91*]. The platform provides a data interface and a set of data modelling tools for the information integration of CIM modules.

Several ***ESPRIT*** projects (*European Strategic Programme for R&D in Information Technology*) concentrate on CIM systems [*Espr92*]. Some of them obtained good results in various aspects, for example, the ***IMPPACT Project*** (*Integrated Modelling of Products and Processes using Advanced Computer Technology*) used the feature-based approach to integrate product and process modelling [*GiBH91, Meie91,Craw93*]; the ***ISA Project*** (*Integrated Systems Architecture for Open Distributed Processing*) developed the *ANSA Platform* (*Advanced Networked Systems Architecture*) which provides an integrated set of structures, functions, design recipes and implementation guidelines for building distributed systems [*Ansa89*]. Perhaps the best conceptual basis and the most advanced architecture is offered within the ESPRIT AMICE projects under the name ***CIM-Open System Architecture*** (***CIM-OSA***). The CIM-OSA concepts are outlined later in this chapter.

1.2) Vendor Concepts of CIM Architectures

This section describes the most advanced concepts of CIM architectures proposed by leading companies, like IBM, DEC and Siemens. At the moment they are not available as complete systems; some elements are on the market, but the other ones are still in development. This section, however, attempts to present the state-of-the-art in this field.

1.2.1) IBM CIM-Architecture

The IBM CIM-Architecture defines an overall structure for information systems which supports information sharing and business process integration for an industrial

enterprise. It focuses on the storage of shared information, its delivery throughout networks and its presentation to application programs, devices and users [*Hug 91*].

The IBM CIM-Architecture is based on a layered structure. It offers layers of software services that provide functions like data management, presentation and communication to the application developer and end user. Figure 1.1 shows the components of the IBM CIM-Architecture [*IBM 91*].

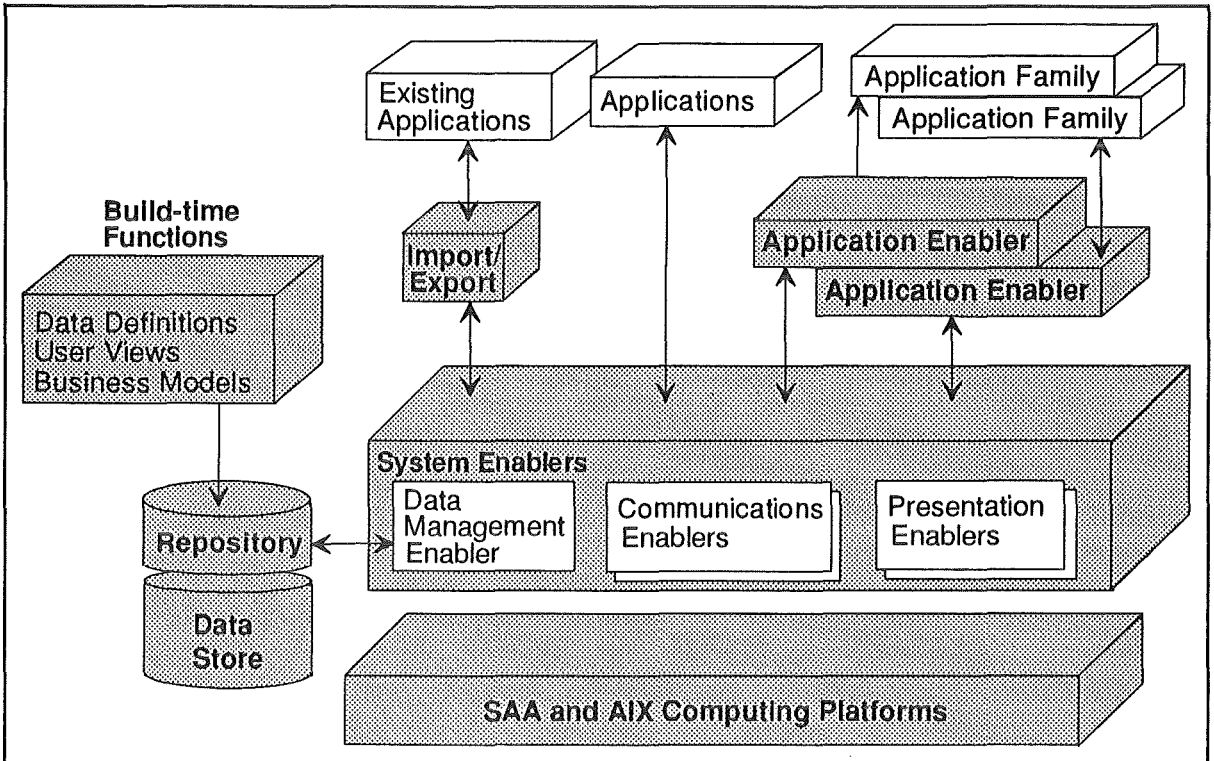


Fig. 1.1 The IBM CIM-Architecture

The IBM CIM-Architecture functions will be implemented on the operating environments of *IBM's Systems Application Architecture* (*SAA*) and *IBM's Advanced Interactive Executive* (*AIX*). SAA is for consistency and compatibility among software products and AIX is for the UNIX environment.

The *data repository* and *data store* are used to manage the enterprise's shared data. The data repository contains a directory of shared data elements, data definitions recognized throughout the enterprise, relationships between data elements, and

data storage locations. The data store is the set of storage facilities containing the shared data.

The *system* and *application enablers* offer application integration functions ranging from generic system management to specific application functionalities. They provide the necessary support for new applications to integrate data and business processes. Three types of *system enablers* are distinguished: 1) *data management enabler* supporting a variety of data repository functions, ranging from file transfer requests to the complex queries, to allow data to be shared easily across the enterprise; 2) *communication enablers* providing a variety of communication protocols and network to allow the interconnection of devices, systems and people; 3) *presentation enablers* providing applications with a device-independent interface to input/output devices such as workstations, industrial computers, sensors, control systems and production equipment. The *application enablers* provide an *Application Program Interface* (*API*) to their application family. They are built on the system enablers.

The *import/export facilities* provide data exchange support through an interface with data management enabler for the integration of existing applications. They extract, transform and communicate data to the data repository and data store.

For the implementation flexibility the IBM CIM-Architecture is used in two environments, *build-time* and *run-time environment*. In the *build-time environment*, data objects, relationships between objects, and business processes are defined. These definitions are stored in the data repository. They are application-independent and therefore can be shared by multiple diverse applications throughout the enterprise. In *run-time*, applications use enablers and the build-time information in the data repository to control application program execution. The run-time environment supports the day-to-day operation of applications, data integration and business process integration.

1.2.2) DEC CIM-Concept

The CIM concept of Digital Equipment Corporation means the improvement of a manufacturing process with the aid of the Information Technology and the integration of the information processing of all enterprise activities [*Flat88*]. It focuses on the integration of applications by information exchange. With top-down approaches to

functionalities of business processes, all the enterprise activities are modelled into several functional blocks. Each functional block may reflect a branch of enterprise activities, such as marketing, sales, budgetting, product design, production and assembly, warehouse or quality control. They are interconnected by an intensive information flow. The DEC CIM model with the information flow can be found in [*Remb90a*].

The DEC CIM concept is realized by the integration of information from multiple applications which act as CIM components. DEC provides a number of CIM components (hardware and software products) for building a CIM system. Each of them can cover one or more branches of the enterprise activities. Figure 1.2 gives the basic outline of a CIM system provided by DEC [*DEC 91*].

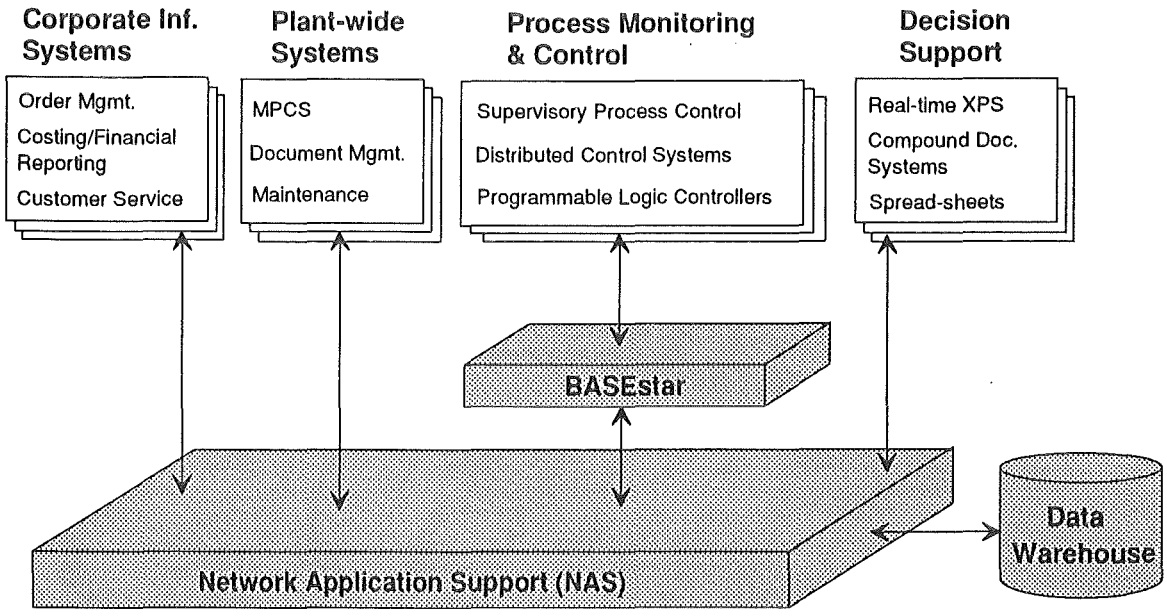


Fig. 1.2 The Concept of DEC CIM-System

In the CIM concept, DEC offers *Network Application Support* (*NAS*) to the manufacturing environment. NAS is an extensive product set designed to integrate applications and information across the manufacturing enterprise. It provides functions to link applications and information from multiple operating systems such as VMS, UNIX, OS/2, VAX-System. NAS is built using the network communication protocol of Ethernet and OSI, such as MAP, X.25, X.400. It provides services to access the shared data stored in Data Warehouse. The Data Warehouse contains a

global directory of shared data definitions and data elements, their locations, etc. for various applications.

NAS environment involves a DEC product, called *BASEstar* which offers software for integrating manufacturing equipment and applications. The important capabilities of BASEstar are: 1) data management for event-driven collection, manipulation, and distribution of plant data, 2) application programming interface for manufacturing integration, 3) ability to control and synchronize application processing, 4) services for controlling and monitoring device operations, as well as for managing the operators and device files, etc. [*DEC 91*].

Corporate Information Systems include order management, costing & financial reporting and customer services. They provide a complete picture of the business through availability of order, customer service and financial information.

Plant-wide Systems include *manufacturing planning & control systems* (*MPCS*), maintenance management systems and document management systems. They allow manufacturing operations to be streamlined. MPCS is used to control material flow throughout the manufacturing process to maintain quality and monitor inventory, while maintenance systems enable manufacturers to reduce down-time and lengthen the life of their production equipment. The document management system provides a timely, efficient, and paperless environment through on-line communication throughout the manufacturing enterprise.

Process Monitoring and Control Systems use the supporting services of BASEstar. They include supervisory process control systems, distributed control systems (DCSs) and programmable logic controllers (PLCs). They monitor and control the production processes to establish consistency in the manufacturing operations, which in turn increases productivity and reduces costs.

Decision Support Tools include three types of systems: real-time expert systems providing continuous guidance to the operators during operation of complex systems; compound document systems; spreadsheets providing ad-hoc reports.

1.2.3) Siemens CIM-Concept

The CIM concept of Siemens is to provide an enterprise with a well-structured strategy for the stepwise integration of all the enterprise subsystems. It gives the

user a guide to realizing an integrated information system for the enterprise. Siemens does not provide a CIM architecture such as IBM or DEC, which serves as a basis for the integration of CIM components. It rather focuses on the grouping of enterprise activities according to their functionality in several domains. Then it goes into well-structured details of each domain and of information flows between these domains. The integration of the domain systems is achieved by the information exchange, via the network systems. Figure 1.3 shows the CIM concept of Siemens [*BaKW89, Remb90a*].

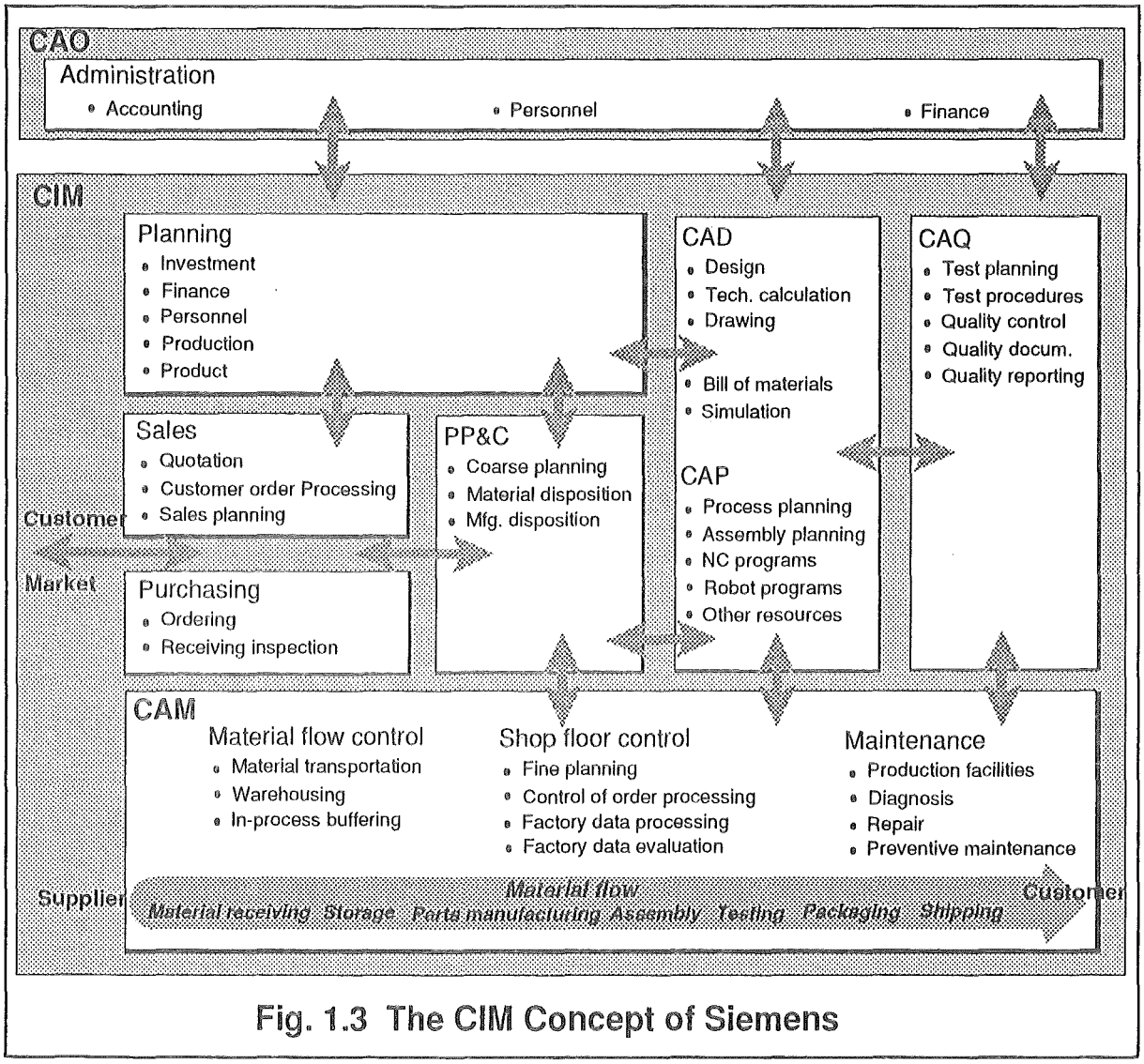


Fig. 1.3 The CIM Concept of Siemens

The concept incorporates a *Computer Aided Organization* (*CAO*) activity which includes accounting, personnel and finance. CIM covers a number of domain

activities such as planning, purchasing, sales, PP&C, CAD, CAP, CAQ, material flow control, shop floor control, and maintenance, etc. Each domain activity is described by the functions and the interfaces to other domains. They are interconnected by intensive information flows.

The distribution of data and the access to the data are of crucial importance for an integrated enterprise-wide data processing system. They will affect the intensity of information flow within a domain system as well as with other domain systems. Therefore, the data should be well classified and distributed in a way that can be easily accessed and maintained. To achieve these functionalities, Siemens applies the concept of hierarchical structure for organizing the domain activities of a manufacturing enterprise in five levels which are namely, Factory, Area, Cell, Station, and Machinery Level. Each level processes data which is mainly stored in its own level. Thus it improves the information flow and reduces the communication loads within and between the hierarchical levels.

Siemens supplies a large set of hardware and software products which can mostly cover the needs of domain activities of a manufacturing enterprise and of communication between the hierarchical levels. The Siemens products can be interconnected to build a 'partial' CIM-system. However, the integration of products from other companies needs sophisticated adaptation work. In recent years, Siemens has actively participated in the international standardization efforts, in order to fufill the increasing user requirements of the integration of CIM-components from different vendors.

1.3) CIM-Open System Architecture (CIM-OSA)

This section describes the CIM-OSA concepts. First it sketches the activities of ESPRIT CIM Projects related to CIM-OSA. Following the basic CIM-OSA concepts, the two main elements, the ***Modelling Framework*** and the ***Integrating Infrastructure*** (***IIS***), will be outlined. The detailed description and analysis of the IIS can be found in [*Hou 92*].

1.3.1) The ESPRIT Projects of CIM-OSA

The ESPRIT program was conceived in 1981 by *the Commission of the European Communities* and European industries. ESPRIT is an industrially oriented R&D

program with the aim of improving the industrial competitiveness of the European Community industries. CIM is an important subject of the ESPRIT programme because the economy of Europe depends heavily on manufacturing industries. Within ESPRIT CIM, key technologies are being developed to address the manufacturing and engineering industries. In order to develop a CIM Architecture in which multi-vendor production system can be implemented at reasonable cost, the ESPRIT project was initiated in 1986 and performed by the ***AMICE*** (reverse acronym for *European Computer Integrated Manufacturing Architecture*) consortium.

The AMICE consortium is the main body which addresses the development of CIM-OSA. It was launched for the ESPRIT Project 688 in 1986 and has been continuing on the consecutive ESPRIT Projects 2422 and 5288. By the end of 1992 the AMICE Projects has spent an amount of about 60 million DM. Figure 1.4 shows the history of the AMICE projects and the deliverables of CIM-OSA documents [*WZL 90*].

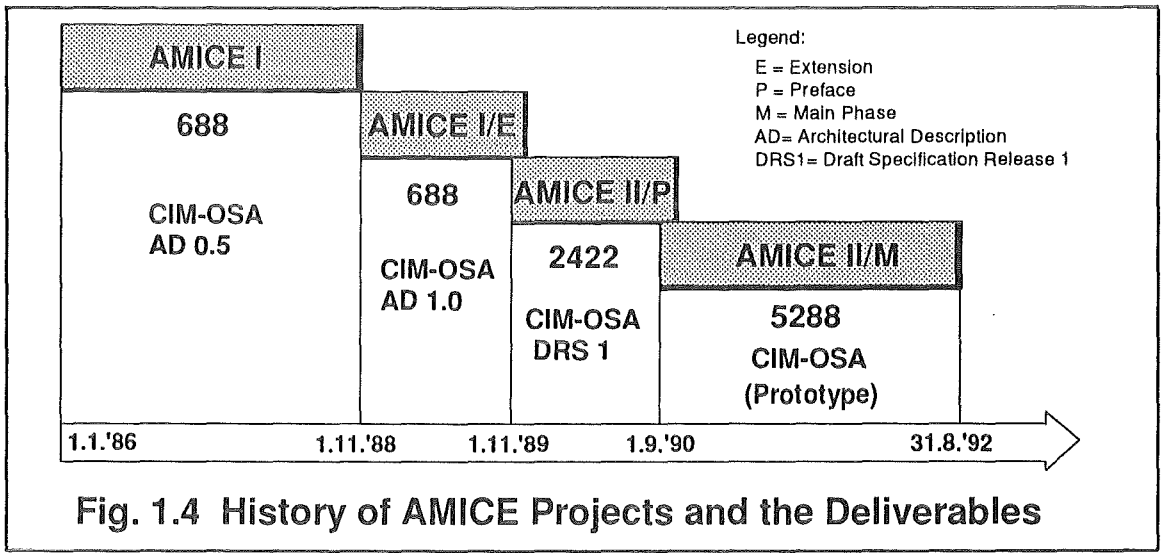


Fig. 1.4 History of AMICE Projects and the Deliverables

The AMICE consortium grouped a changing number of participants, for example in 1989 it consisted of 21 companies from 7 European countries. CIM users, CIM implementors, software houses and research institutions were represented [*AMIC89*]. These were CAP Gemini SESA (Belgium); Procos (Denmark); AEG, DEC, Dornier, IBM, Siemens, Volkswagen, WZL-Aachen Uni. (Germany); Aerospatiale, Alcatel, Bull, Hewlett-Packard (France); FIAT, Italsiel, SEIAF (Italy); APT Nederland BV, Philips (Netherlands); British Aerospace, GEC, ICL (United Kingdom).

The main goal of the current AMICE II/M Project is to develop, validate and publish a first set of CIM-OSA functional specifications (CIM-OSA Release 1). This Release 1 will be validated through prototyping and real operation of relevant parts of CIM-OSA. These prototypes will also be used for demonstrating the capabilities of CIM-OSA. Figure 1.5 shows the AMICE II/M Project workplan [*AMIC90*].

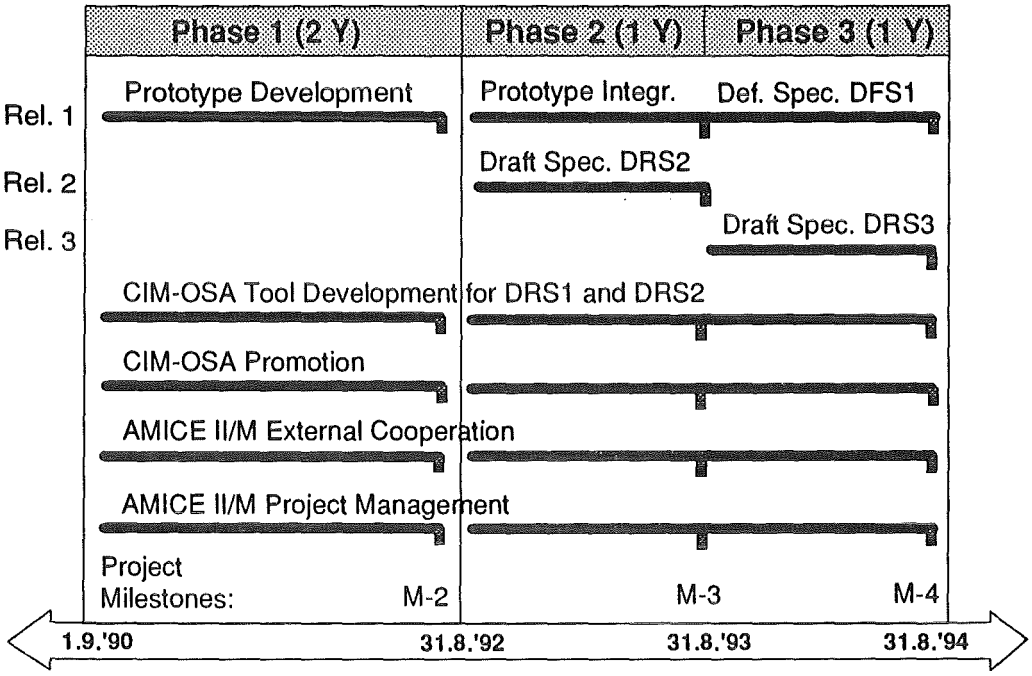


Fig. 1.5 AMICE II/M Project Workplan

In parallel to the AMICE Project, two other ESPRIT projects have been started in 1991 for the validation of the CIM-OSA concepts. One is the ESPRIT Project 5510 ***VOICE*** (*Validating Open System Architecture in Industrial CIM Environment*). Another one is the ESPRIT Project 5499 ***CODE*** (*Computer-supported enterprise-wide Data Engineering*).

The VOICE Project is to validate the CIM-OSA concepts in three industrial sites: a car manufacturing plant, a part manufacturing plant and a casting plant [*VOIC90*]. VOICE intends to demonstrate the CIM-OSA on two testbeds by end of 1992. From the first analysis result to the current state of IIS, it is recognized that the implementation of IIS is not possible. It is not only because of the programming manpower, but mainly due to the underdeveloped IIS. A so-called Special Interest Group VOICE-AMICE Cooperation (SIG-VAC) formed by both consortia is therefore

called upon to cooperate and further develop the IIS. In order to enable the demonstration of CIM-OSA IIS in VOICE testbeds, the current activity in VOICE Project is focused on searching for existing software products which are able to cover the necessary IIS functionality to some realistic extent.

The CODE Project aims at supporting the process of enterprise-wide data engineering in all phases of the system life-cycle [*CODE90*]. It deals mainly with the concepts of the CIM-OSA instantiation process for the information view. The project intends to detail the methodologies for the creation of 'information' reference models and for the instantiation of these reference models into the particular models of a certain enterprise. It will evaluate the constructs of information view given by CIM-OSA, as well as the SCHEER's reference models [*Sche89, Hars92*] by use of the tool set developed by the Manager Software Productions GmbH. In the project, two testbeds will be utilized for the implementation of all tools and functionalities developed by CODE.

1.3.2) The CIM-OSA Concepts

CIM-OSA defines an integrated methodology to support all phases of a CIM system life-cycle from requirements specification, through system design, implementation, operation and maintenance [*AMIC89*]. It provides a ***Modelling Framework*** and an ***Integrating Infrastructure*** (***IIS***). The Modelling Framework supports the modelling of business activities of an enterprise. The IIS is an operating infrastructure supporting the execution of CIM-OSA models and the integration of heterogeneous systems. With both Modelling Framework and Integrating Infrastructure, CIM-OSA enables a consistent and complete information processing from the process design to the manufacturing. Thus it enables the enterprises to perform their business in a real time adaptive mode. Serveral papers appeared to introduce the basic concepts of CIM-OSA [*Kosa90, Klit90-91, Pans90, Beec90, JoVe90, Vern90, Quer91*].

The spirit of CIM-OSA may become obvious, by first identifying the problems and requirements of current CIM-applications, and then introducing the CIM-OSA concepts. Figure 1.6 depicts the software development of a CIM-application in a current industrial environment. The task of software engineering can be globally divided into four subtasks: requirements definition, design specification, implementation and testing. The whole software engineering task is an iterative approach. A review or modification of previous subtasks is often needed in order to

achieve a desired result in the current subtask. After sufficient testing and validation, the software product will then be released.

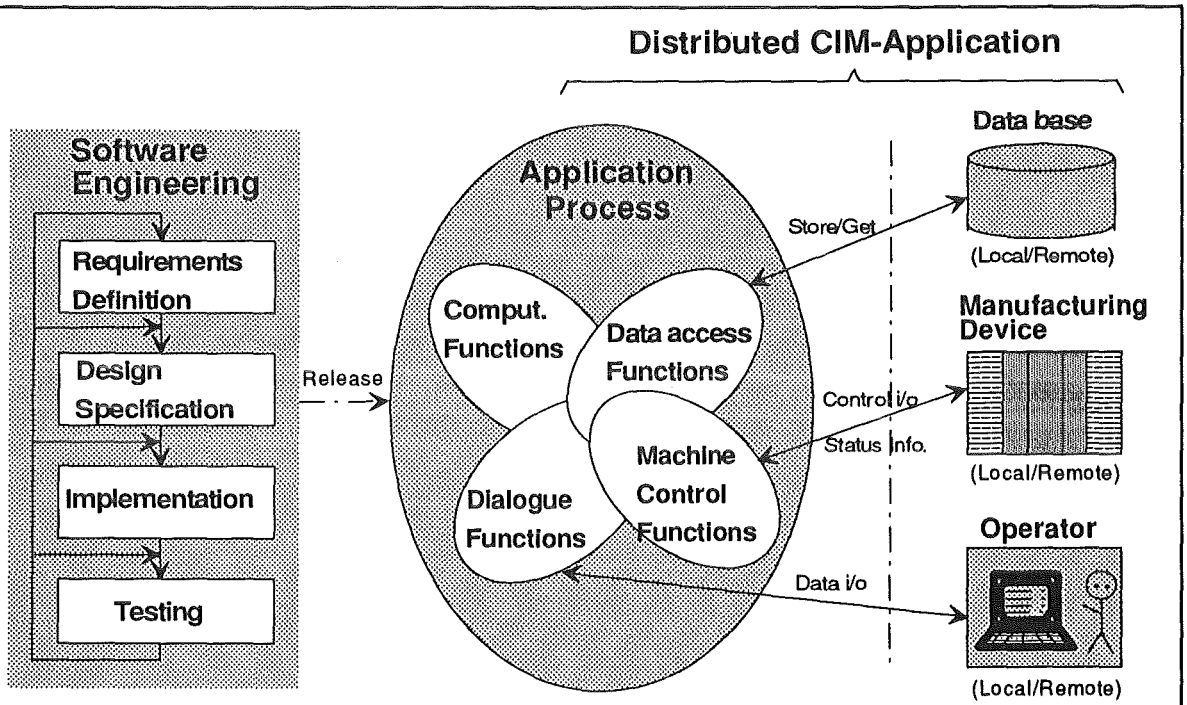


Fig. 1.6 Current Software Development of a CIM-Application

In this kind of released software program, however, not only the ***Task Order Sequence*** is embedded, but also a large number of *functions* are included. The *Task Order Sequence* describes the process order of the application. The functions can be classified into four types: 1) *computational functions* for arithmetical operations; 2) *data access functions* for storing and getting data from the data bases, 3) *machine control functions* for sending the control signals to and getting machine status information from manufacturing devices; and 4) *dialogue functions* for interacting with operators. A function may also include all or some of these functionalities.

The current CIM-applications contain a large number of different types of functions. These functions are used for the data exchange with the vendor-dependent devices, e.g. manufacturing machines, storage media, display and input devices. Therefore, any change of devices in an enterprise environment needs a modification of the application. It is well-known that the adaptation of an existing application to the new environment is mostly very time-consuming and costly.

If a CIM-application is used for an environment in which the required functions and the operating devices are very stable for a long period, problems will not be met. However, in todays situation, particulary in the manufacturing area an enterprise can keep in competition with its products, only if it is able to adapt its manufacturing resources immediately to a changing environment. From the CIM viewpoint, besides the data communication, the following new requirements have to be fulfilled:

- the *Task Order Sequence* should be kept flexible such that it can be reconfigured by the CIM user;
- the *functions* should be implemented as program units which can be loaded, removed, or started according to the user needs;
- the CIM user should be supported, by describing the *Task Order Sequence* as well as the functions which the user requires.

These requirements clarify that the *Task Order Sequence* of a CIM-application should be separated from the *functions* and should be configurable for the CIM user. In order to fulfil the requirements described above, CIM-OSA provides a Modelling Framework and an Integrating Infrastructure which are outlined in the following sections. Moreover, CIM-OSA addresses also the integration of 'islands of automation' and the interoperability/portability of vendor-dependent devices.

1.3.2.1) The Modelling Framework

The Modelling Framework is known as the CIM-OSA cube shown in Figure 1.7 [*AMIC89,Beek90,JeVo90,Vern90,Vlie90*]. It provides a *Reference Architecture* which *Particular Architectures*, covering the needs of individual enterprises, can be instantiated from.

The *Reference Architecture* includes two architecture levels, called *Generic* and *Partial Level*. It provides a set of generic building blocks, partial models and user guidelines for each of the three modelling levels (requirements definition, design specification and implementation description). The partial models are used to reduce the modelling effort and to increase the portability of models for a particular enterprise. They are applicable to one or more industrial sectors, such as automotive manufacture, machine tool, electronic, aerospace, etc.

The *Particular Architecture* is provided for modelling a particular enterprise, i.e. it exhibits a given CIM solution. It contains the requirements for the specific enterprise with all its system components. The contents of the Particular Architecture can be instantiated from those of the Reference Architecture.

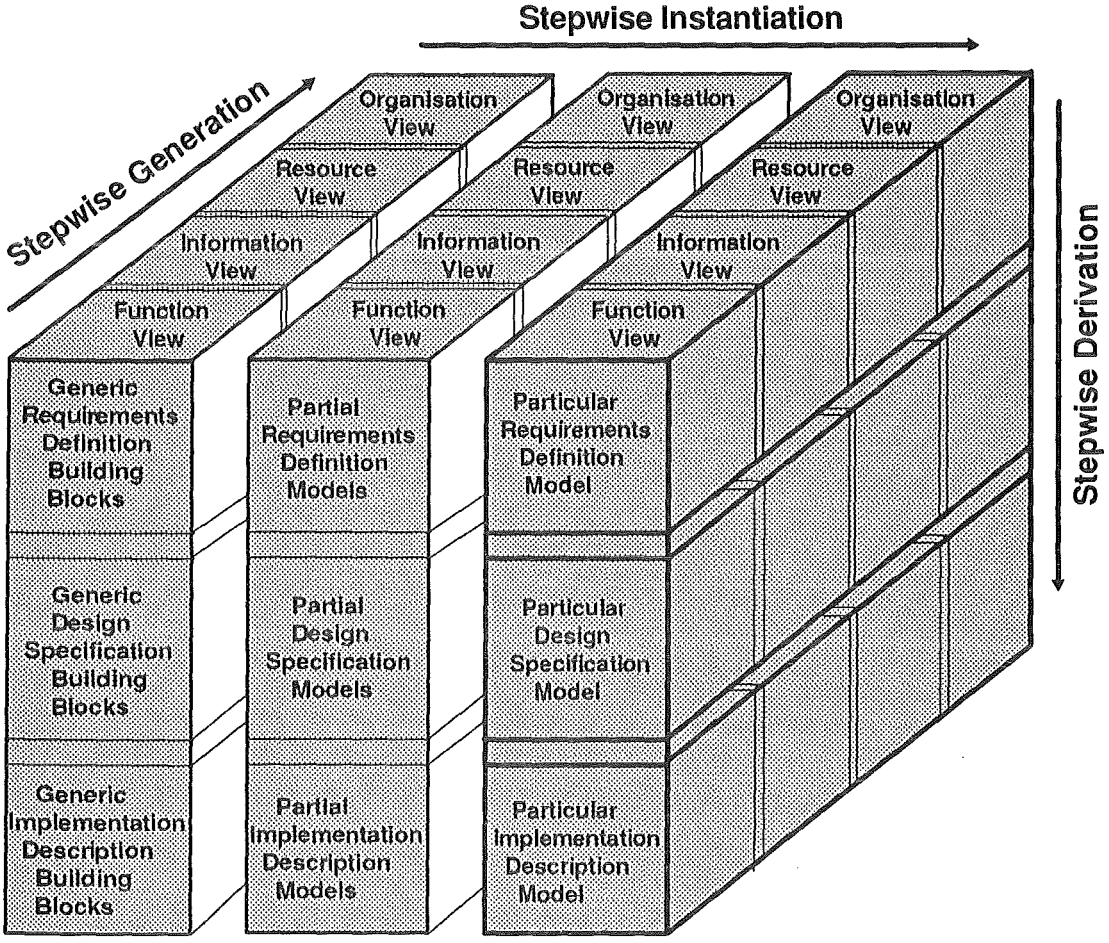


Fig. 1.7 The CIM-OSA Architecture Framework

For each modelling level, CIM-OSA provides 4 different views to describe the enterprise activities. These are: 1) ***Function View*** for decomposing functions of a Domain Process; 2) ***Information View*** for describing information objects that are used to carry out the functions; 3) ***Resource View*** for describing the available resources in order to optimize them regarding the integration requirements; and 4) ***Organization View*** for charging the responsibilities of organizational entities for the execution of domain processes and functions.

CIM-OSA describes the enterprise activities by the *constructs* (*building blocks*) of ***Enterprise Function*** (***EF***). The Enterprise Function is the generic construction used for the description of the enterprise at each level of functional decomposition, such as for the Domain Process, Business Process, Enterprise Activity, or Functional Operation defined below. It provides a uniform way of defining functionality, behaviour and functional structure of any CIM-OSA domain. Figure 1.8 shows the definition of an Enterprise Function [*Klit91b*] which is described in two parts:

a) a *Control/Behaviour Structure* consisting of more elementary Enterprise Functions, Procedural Rules and a set of structural links. The ***Procedural Rules*** dictate the behaviour of the structure by the use of conditions.

b) a *Transformation Function* consisting of input/output information described by a list of object views.

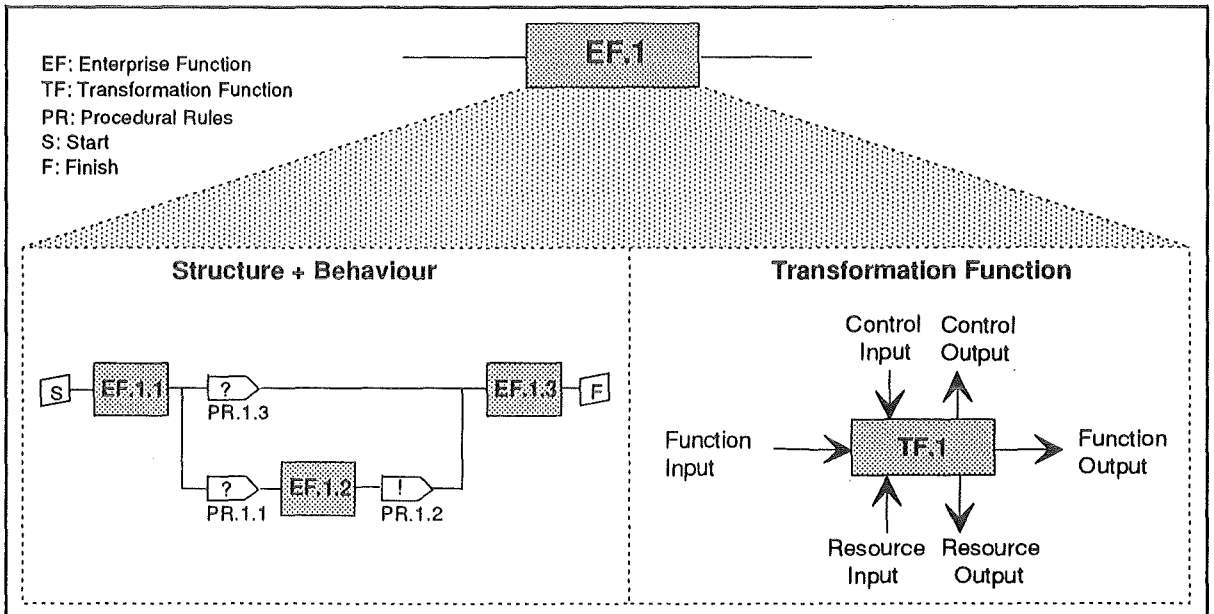


Fig. 1.8 Definition of an Enterprise Function

The functional decomposition is described in a top-down hierachical structure. Figure 1.9 shows a tree of functional decomposition. The uppermost level of the tree is called *Domain Process*. A ***Domain*** is a selected working area for the achievement of a specified task within an enterprise. It can be for the description of all the activities of a manufacturing cell. It can also be for a business goal like 'Make Profit'. A *Domain Process* contains the task description of the domain, including the running procedure and information.

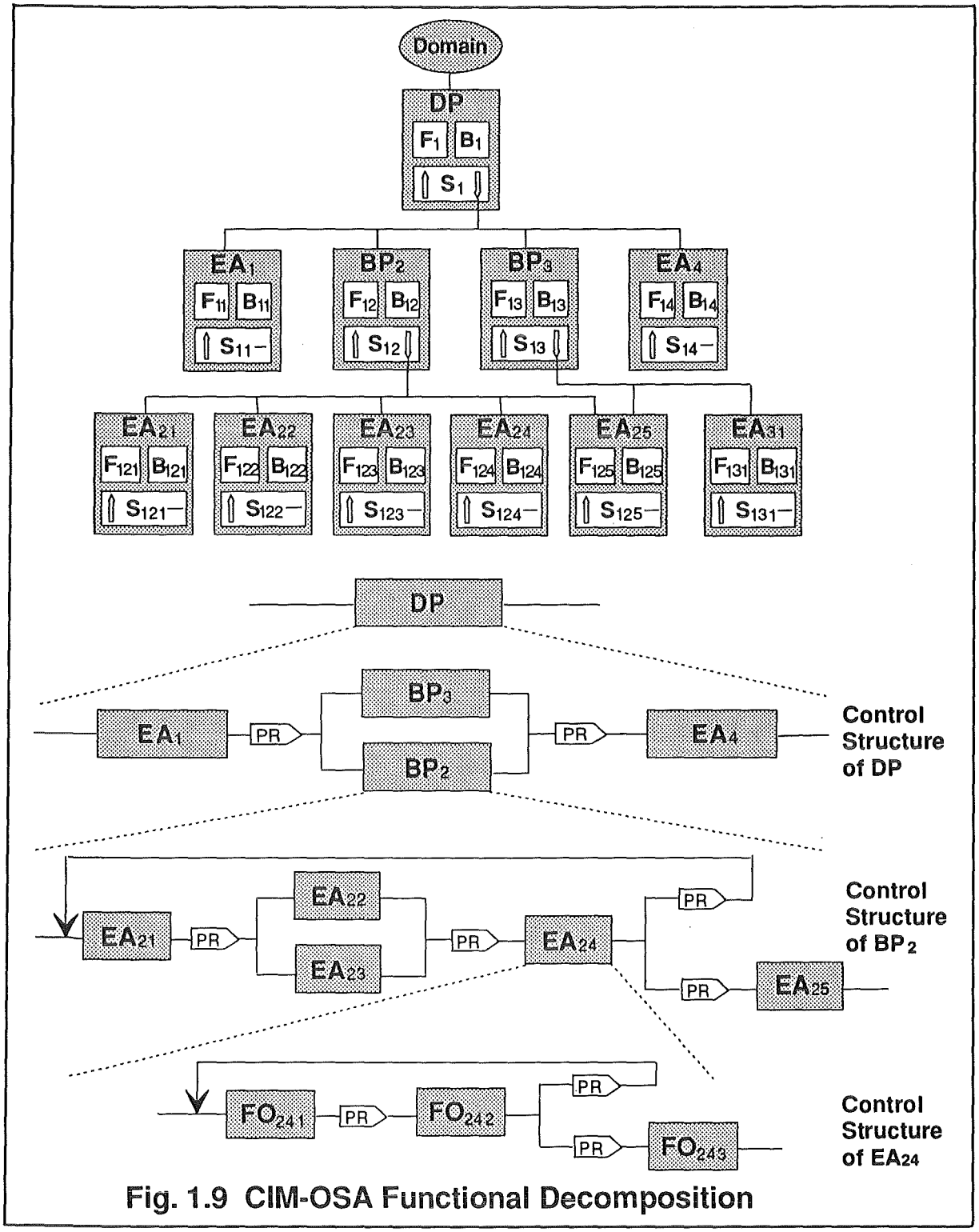


Fig. 1.9 CIM-OSA Functional Decomposition

Keys are: DP : Domain Process, BP : Business Process, EA : Enterprise Activity,
F : Functionality, B : Behaviour, S : Structure,
PR : Procedural Rule, FO : Functional Operation.

A ***Domain Process*** can contain a number of ordered Business Processes or Enterprise Activities. A ***Business Process*** can be further decomposed into more elementary Business Processes, or can consist of Enterprise Activities. An ***Enterprise Activitiy*** is defined by a set of ***Functional Operations.*** The tree of the CIM-OSA functional decomposition (cf. Fig. 1.9) represents the *Task Order Sequence* which is embedded in a traditional application process as shown in Fig. 1.6. This however in CIM-OSA is extracted from the application process, and therefore can be defined and reconfigured by the modeller in a more flexible way.

Functional Operation (FO) **is considered as the basic elementary CIM-OSA Function Model.** Within the Modelling Framework, a *Functional Operation* is defined as a basic controllable unit of information processing which is not required to be further decomposed. It is for accomplishing a specified job, e.g. mounting screws on a car body. The CIM-OSA modelling methodology consists of splitting *Functional Operations* into three categories which are called ***Application***, ***Human*** and ***Machine Functional Operation*** (***AFO***, ***HFO*** and ***MFO***).

Figure 1.10 shows a Domain Process Model which is designed for a simple master-slave control system in a manufacturing cell. It reads the path coordinates of a master robot driven by an operator and then sends them to a slave robot. With the received path coordinates the slave robot follows its path accordingly. The Domain Process Model has only one Business Process Model. The Business Process Model contains five Enterprise Activities in an order of consequence. The Procedural Rules are constructed by predefined objects. For the sake of simplification, each Enterprise Activity contains simply either a single Functional Operation or two Functional Operations in parallel.

The function model 'Control' of MFO-30 is used to read the path coordinates of the master robot and to send them to the slave robot. With the received path coordinates the slave robot moves along its path accordingly. The 'Monitoring' of MFO-40 is used to read the path coordinates of both robots and put the path deviations into a database for the analysis. MFO-10 and MFO-50 are used to set up and release both robots respectively. The HFO-20 is provided for the human intervention. An operator can stop, restart or terminate the activities of 'control' and 'monitoring'. This Domain Process Model is used for the testing of the MF-Prototype, which is described in detail in Chapter 7.

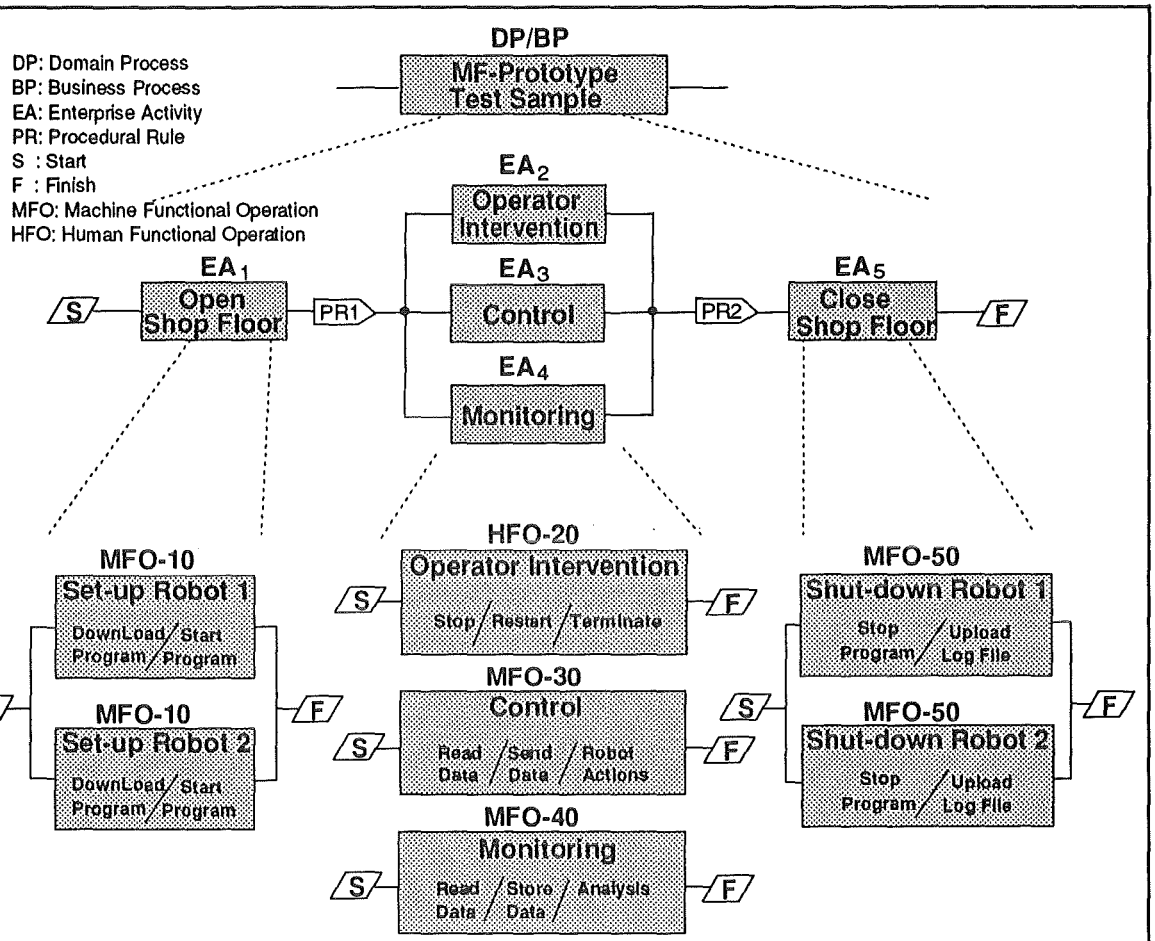


Fig. 1.10 A Domain Process Model for the Test Scenario

Figure 1.11 shows the main feature of CIM-OSA. It provides a link between the process design and the manufacturing. After the modelling task, a number of *Particular Enterprise Models* (i.e. *Domain Process Models*) associated with information/resource objects are released for a particular enterprise. They are stored as CIM-OSA data in external storage media and can be thereafter started by an operator.

CIM-OSA is involved in two environments: ***Integrated Enterprise Engineering*** and ***Integrated Enterprise Operation*** environment. The Integrated Enterprise Engineering environment is a build-time environment in which CIM-OSA provides the CIM-user with the Modelling Framework to define their business activities. The Integrated Enterprise Operation environment is a run-time environment in which the Integrating Infrastructure uses the information objects to control application program execution. The Integrating Infrastructure is described further in the next section.

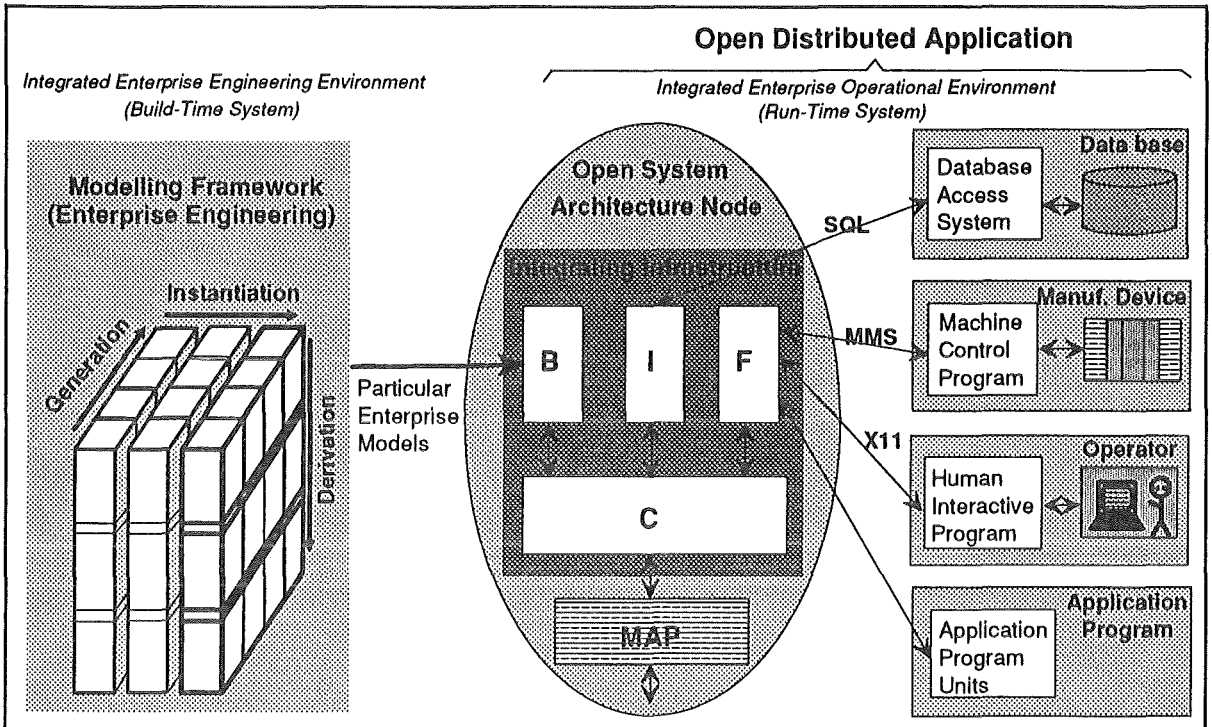


Fig. 1.11 CIM-Open System Architecture

1.3.2.2) The Integrating Infrastructure (IIS)

The ***Integrating Infrastructure*** is an operating infrastructure supporting the execution of CIM-OSA models [*AMIC89,Klit90-91,Quer91*]. It is a software program which may include several modules to achieve its specified functions. Depending on the configuration of a given CIM-OSA system, the software modules of the Integrating Infrastructure can be installed on several network stations. Figure 1.12 depicts the main frame of the Integrating Infrastructure which comprises four blocks of services: ***Business***, ***Front-End***, ***Information*** and ***Communication Services*** [*WZL90*]. Their functionality and content is briefly described below.

Communication Services (C-Services)

The ***Communication Services*** serves as a bridge between the other IIS components and the underlying communication subsystems. It is used to enable the other IIS components to cooperate system-wide, while isolating them from the underlying communication subsystems; it provides functions for location, access and performance transparency. CIM-OSA deals with all the communication facilities, not

only for the OSI networks, but also for proprietary communication networks. The current Communication Services consists of two elements [*CIM-OSA 90/C5-1000, Moli92*], namely:

- ***System Wide Exchange*** (***SE***) which provides a set of callable functions for message-passing and service administration. It offers synchronous and asynchronous communication. It handles the intra-node communication and allows inter-node communication by means of the underlying Communication Management.
- ***Communication Management*** (***CM***) which provides functions to manage the use of underlying communication service for the inter-node communication through networks. It provides a transparent access to the underlying communication subsystems and manages the communication resources.

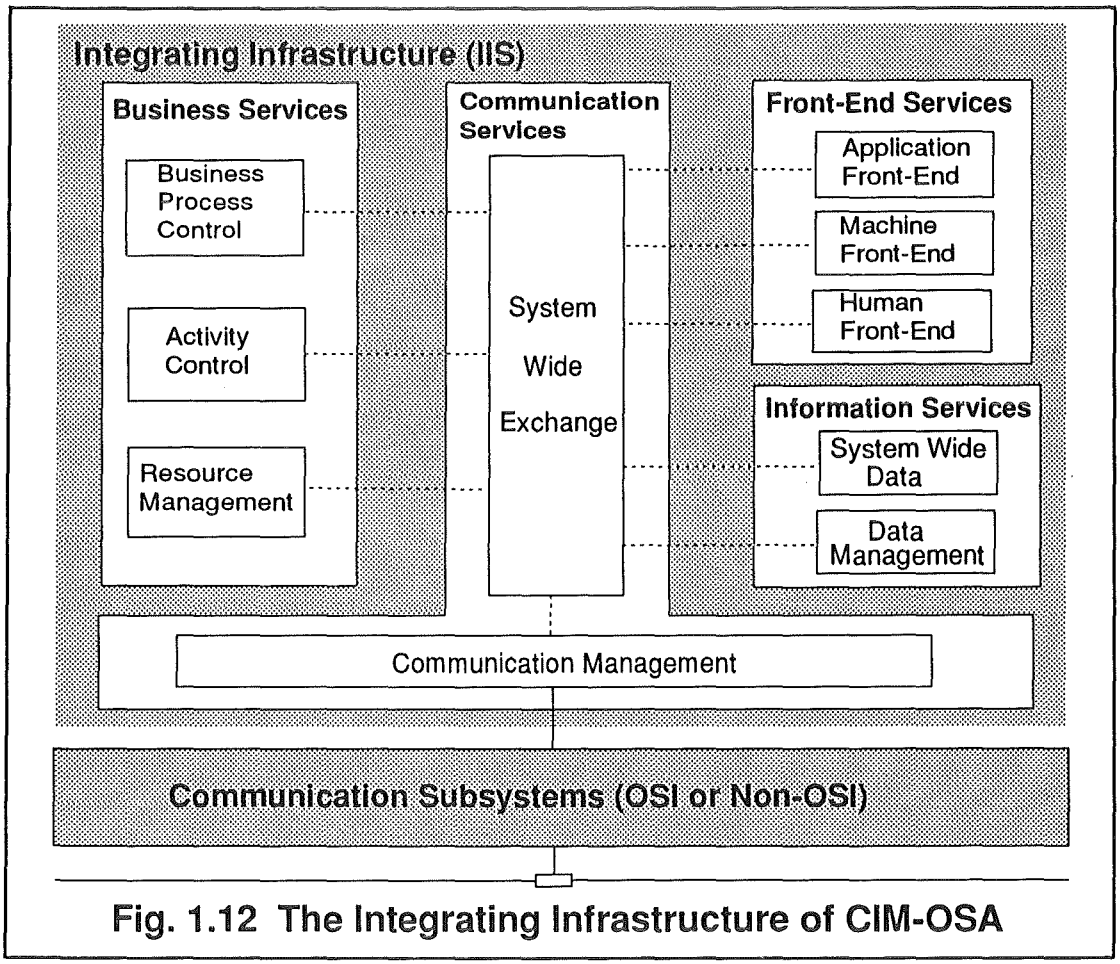


Fig. 1.12 The Integrating Infrastructure of CIM-OSA

Business Services (B-Services)

The ***Business Services*** provides functions required to control the execution of the CIM-OSA models (i.e. Domain Process Models). It deals mainly with the results of the *Function View* and *Resource View* of the CIM-OSA modelling task, i.e. it manages the occurrences (instances) of Business Processes and Enterprise Activities, schedules the manufacturing resources and dispatches the Functional Operations to the Front-End Services for the control of their execution. The Business Services includes three elements [*CIMO90/C5-4000*], namely:

- ***Business Process Control*** (***BC***) which manages and dispatches the execution of business processes and enterprise activities by interpreting the Procedural Rules;
- ***Activity Control*** (***AC***) which controls the execution of Enterprise Activities by dispatching the Functional Operations which are carried out by the enterprise resources under the control of the Front-End Services;
- ***Resource Management*** (***RM***) which reserves and dynamically schedules the enterprise resources.

Figure 1.13 depicts the content of Business Services with the working process. The processing of a Domain Process Model can be described as follows:

1) a Domain Process Model can be started by a human operator. A Domain Process Model may consist of several Business Processes (BPs);
2) BC creates BP-occurrences (instances) and requests RM to schedule the BP-occurrences;
3) RM creates a schedule for the BP-occurrences. The schedule is designed by use of behavioural structure (task flow control) of the BP-occurrences. Each BP-occurrence may consist of a number of Enterprise Activity (EA) occurrences such that each of them in turn may contain several ordered Functional Operations (FO's);
4) BC checks the schedule. When the input condition of an EA in the schedule table is met, BC will ask AC to execute the EA;
5) AC controls the processing of each FO involved in the EA;
   5-1) AC asks RM to assign Resource Units for the first FO of the EA;
   5-2) RM assigns Resource Units to the FO;
   5-3) AC requests the Front-End Services to execute the FO;

5-4) After reception of the FO response from Front-End Services, AC starts the next FO (if any). After the last FO of the EA has been completed, AC reports this to BC;

6) BC acts on the status of the EA and checks next EA of the BP-occurrence (repeats the points 4 and 5). When the last EA is completed, BC acts on the status of the BP-occurrence and continues to start the next BP-occurrence.

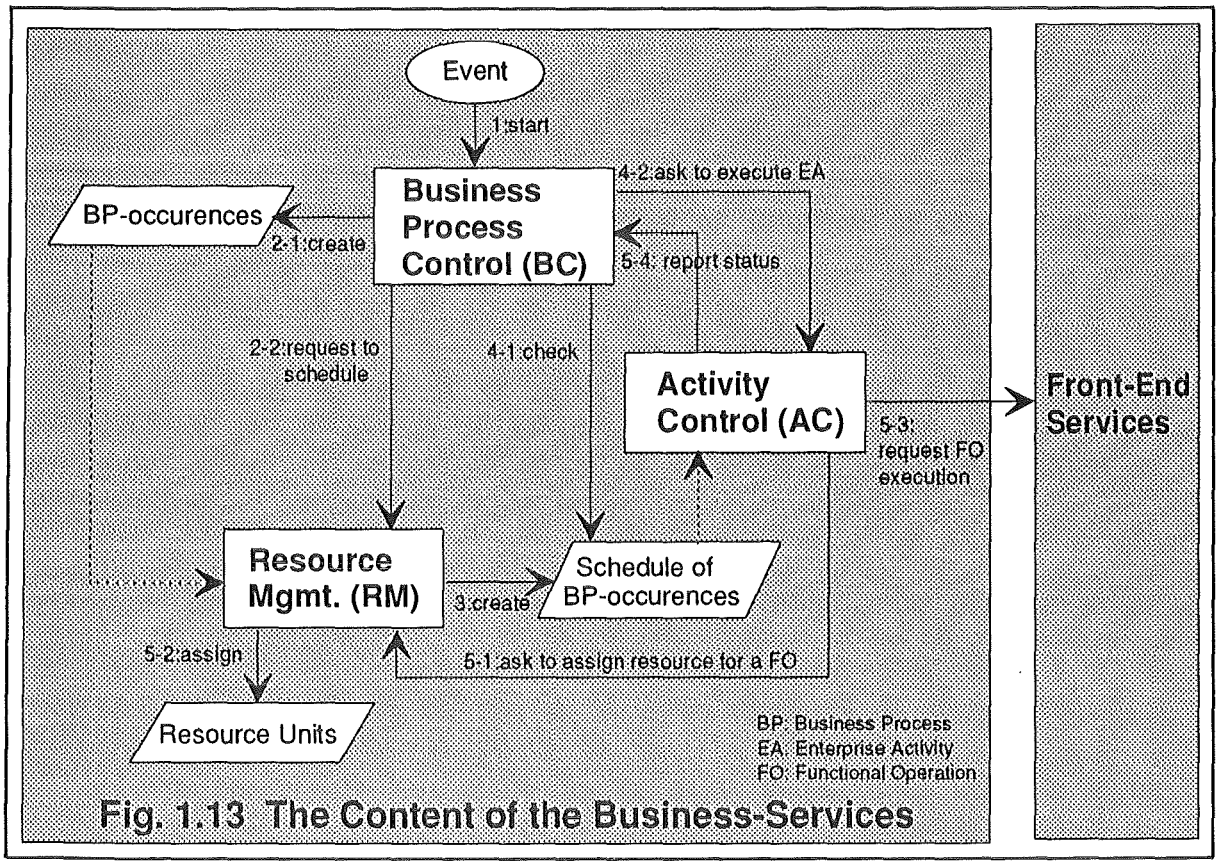


Fig. 1.13 The Content of the Business-Services

Information Services (I-Services)

The ***Information Services*** provides functions required to access all the CIM-OSA distributed data in a unified way. It integrates existing data storage systems, such as Database Management System or File System, and accomodates fundamentally different data structures whilst presenting them in a unified way. The Information Services deals mainly with the results of the *Information View* of CIM-OSA modelling. It is designed as a service which provides only a set of callable functions for its

users. The Information Services consists of two elements [*CIMO90/C5-2000*]. These are:

- ***System Wide Data*** (***SD***) which coordinates and transforms the data requests sent by the other IIS services in CIM-OSA format into *System Query Language* (*SQL*)-format. It provides its users with a unified access to data without any concern for the data location and the data storage structure, and without taking care of all processing associated with the distributed nature of the accessed data (e.g. replication, difference in schemas, system-wide consistency enforcement and access rights enforcement).
- ***Data Management*** (***DM***) which translates the data requests in SQL-format into the access language of a particular DBMS or File system. It provides its unique client, SD, with functions for storage, retrieval and conversion of data.

Front-End Services (F-Services)

The ***Front-End Services*** takes, on one side, the control and management of execution of *CIM-OSA elementary Function Models*, and on the other side, integrates the heterogeneous world of enterprise resources. It deals with the problem of interfacing the vendor-specific applications. However it is not only simply an interface between other IIS components and the external CIM-Modules. Moreover, it should take the control of function execution to some extent.

In CIM-OSA, three types of enterprise resources (external CIM-Modules) are distinguished: machine control programs, application programs and human interactive programs. These enterprise resources are referred to as ***Application***, ***Human*** and ***Machine Functional Entity*** (***AFE***, ***HFE*** and ***MFE***). They are controlled by the three corresponding elements of Front-End Services[*CIM-OSA90/C5-3000*]:

- ***Application Front-End Services*** (***AF***) provides functions for interfacing pure data-processing applications. It has to integrate all application-specific data processing functions such as CAD (Computer Aided Design), CAM (Computer Aided Manufacturing), PPC (Production Planning & Control), PMS (Production Management System), stock management, computer aided maintenance, etc.;
- ***Human Front-End Services*** (***HF***) provides functions for mediating between the IIS and its human users. CIM-OSA IIS provides the HF to integrate the human work by separating the dialogue component from the actual application programs. The HF supports services for the device-independent part of this dialogue

component and therefore provides an environment in which good user interfaces can be constructed, implemented, executed, modified, evaluated and maintained;

- ***Machine Front-End Services*** (***MF***) provides functions for integration and control of manufacturing devices. It has to integrate machine specific functions such as robots, NC-machines, programmable control applications, etc.

Figure 1.14 gives an overview of the Front-End Services environment and its internal relationships. The dotted lines indicate only the virtual connections.

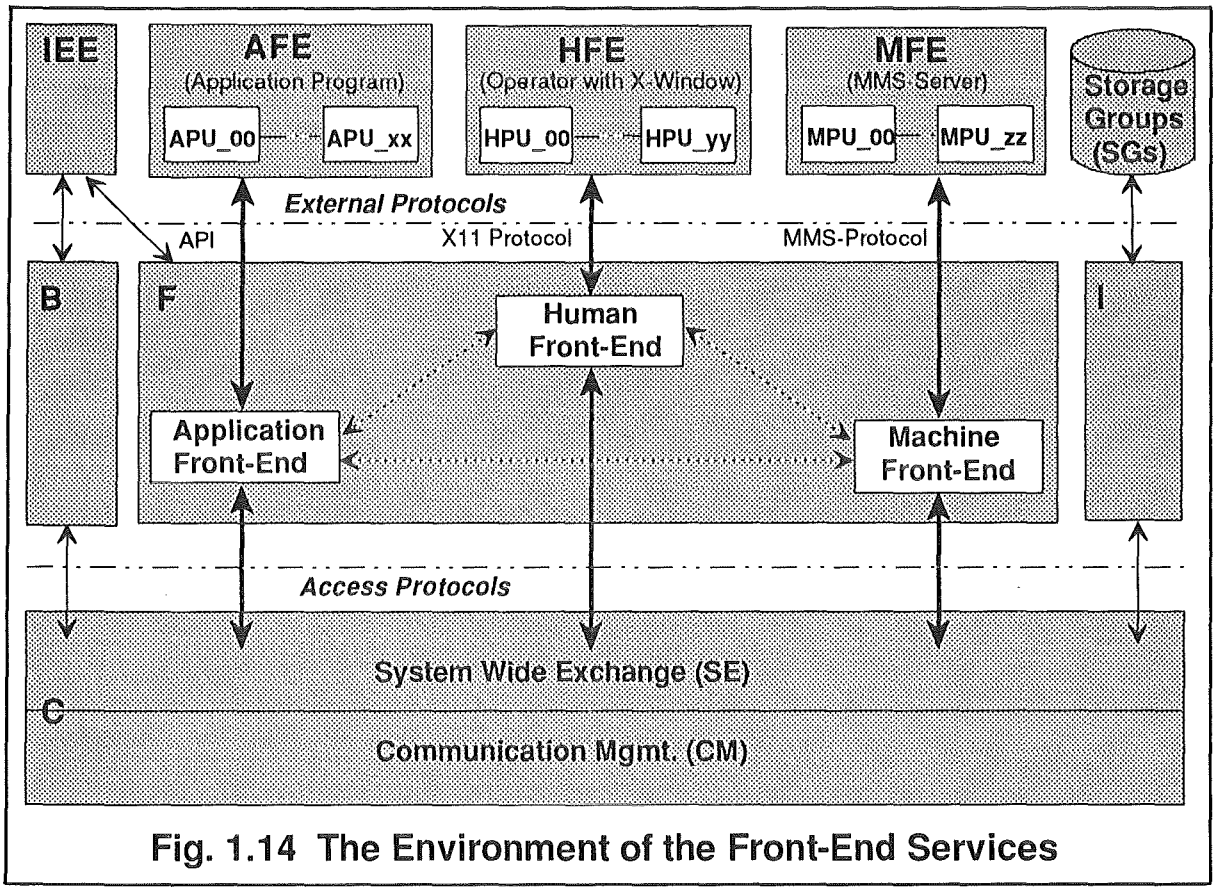


Fig. 1.14 The Environment of the Front-End Services

Keys are:

IEE: Integrated Enterprise Engineering,	
AFE: Application Functional Entitiy	APU: Application Program Unit,
HFE: Human Functional Entity,	HPU: Human Program Unit,
MFE: Machine Functional Entity,	MPU: Machine Program Unit,
MMS: Manufacturing Message Specification,	API: Application Program Interface.
B : Business Services,	F : Front-End Services,
I : Information Services,	C : Communication Services.

In the CIM-OSA world, the information exchange between the IIS components are by use of callable functions of the System Wide Exchange. Three types of Protocols defined for Front-End Services are distinguished: *Access*, *External* and *Agent*

Protocol. The ***Access Protocol*** is used by other IIS services to send and receive message to/from the Front-End Services. The message unit is called *Protocol Data Unit*. The content of *Protocol Data Unit* depends on the services supported by the three elements of Front-End Services. The ***External Protocol*** is used to co-operate with external CIM-Modules. CIM-OSA intends to use the international or de-facto standards as External Protocol, e.g. MMS-protocols, X/Open Protocols, standard Application Program Interface. The ***Agent Protocol*** is used for the co-operation between two of the same elements of Front-End Services, e.g. between two Machine Front-Ends, installed on different network nodes. The *Agent Protocol* is not presented in Figure 1.14.

The external CIM-Modules (i.e. AFE, HFE and MFE) consists of a number of *Program Units* which are called *Application, Human* and *Machine Program Unit* (*APU*, *HPU* and *MPU*), respectively. A ***Program Unit*** is an application-dependent program function which performs the functions of its corresponding FO. The call of a *Program Unit* can be achieved by means of software mechanisms such as a procedure call (remote or local), the execution of a process or of a set of processes, sending a message to a mailbox, or sending a message to a pipe or to a queue, etc.

All three elements of Front-End Services have very similar functions. Each of them provides for the control and management of one of the three types of FO's. As stated before, FO's result from the Domain Process' hierarchical decomposition of CIM-OSA modelling framework. At the *Function View* of the *Design Specification Modelling Level*, CIM-OSA provides a construct for the description of FO.

For the execution of a FO, the Front-End Services will call the appropriate *Program Unit* installed in the external CIM-Modules. As stated, the external CIM-Modules are viewed as enterprise resources required to perform FO's. And they are described by constructs of the *Resource View* of the *Design Specification Modelling Level*.

The content of each type of external CIM-Modules is futher shown in Figures 1.15, 1.16 and 1.17. They sketch the position of each Front-End Services element and its co-operating partners. Each of them should be able to manage several CIM-Modules on behalf of other IIS services. Figure 1.15 shows that the ***Application Front-End (AF)*** presents to other external CIM-Modules (i.e. AFE's) a standard *Application Program Interface* (*API*). An *Application Functional Entity (AFE)* can be implemented as a set of procedures, a process under the control of an operating system, or a set of (local, remote or distributed) communicating processes.

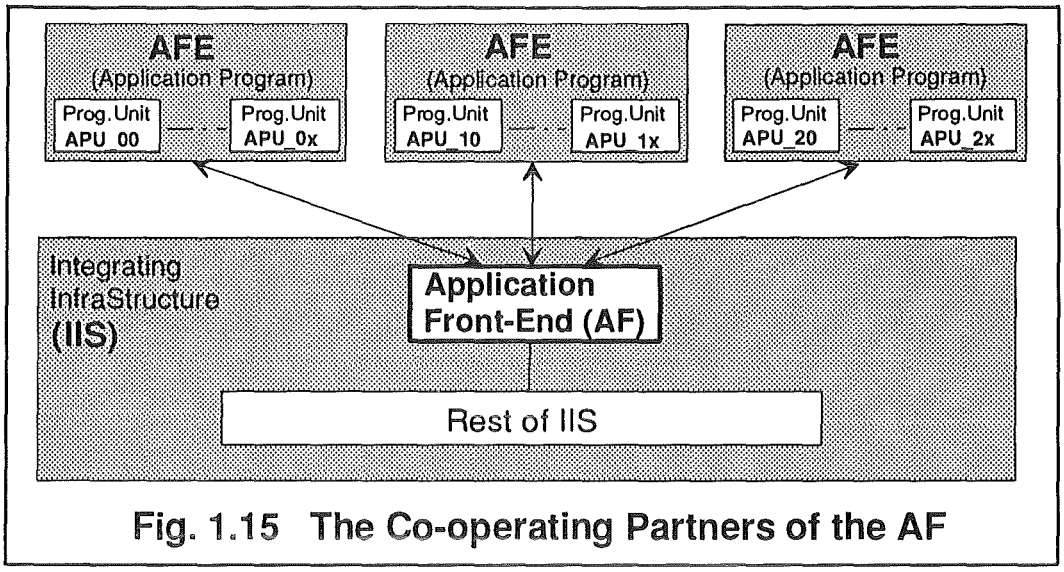


Fig. 1.15 The Co-operating Partners of the AF

The ***Human Front-End (HF)*** presents to other external CIM-Modules (i.e. HFE's) a standard interface, e.g. X/Open. A *Human Functional Entity (HFE)* is comprised of the operational human user, the interaction device, the device control (device drivers), and the Device Agent. A *Device Agent* consists of a number of Program Units. It performs as far as necessary the network services, protocol conversions between the HF-External Protocol and the special device protocols, etc.

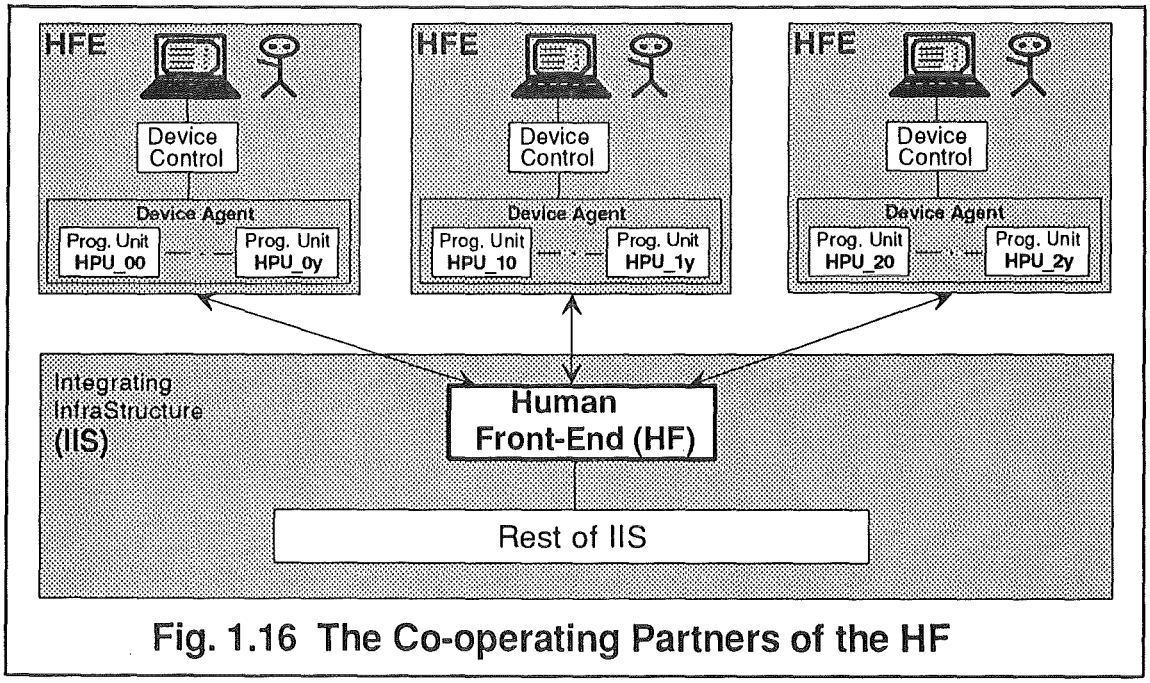


Fig. 1.16 The Co-operating Partners of the HF

The ***Machine Front-End (MF)*** presents a uniform protocol to the various types of external CIM-Modules (i.e. MFE's) such as robot controllers, numerical controllers, programmable logic controller, process monitoring, control systems, etc. This protocol is called MF-External Protocol which will be based on the international standard of *MMS* (*Manufacturing Message Specification*). A *Machine Functional Entity (MFE)* is comprised of the machine physical device, the machine specific device controller, and the machine control program. In case of MMS-devices, the machine control program will use the MMS supported services primitives.

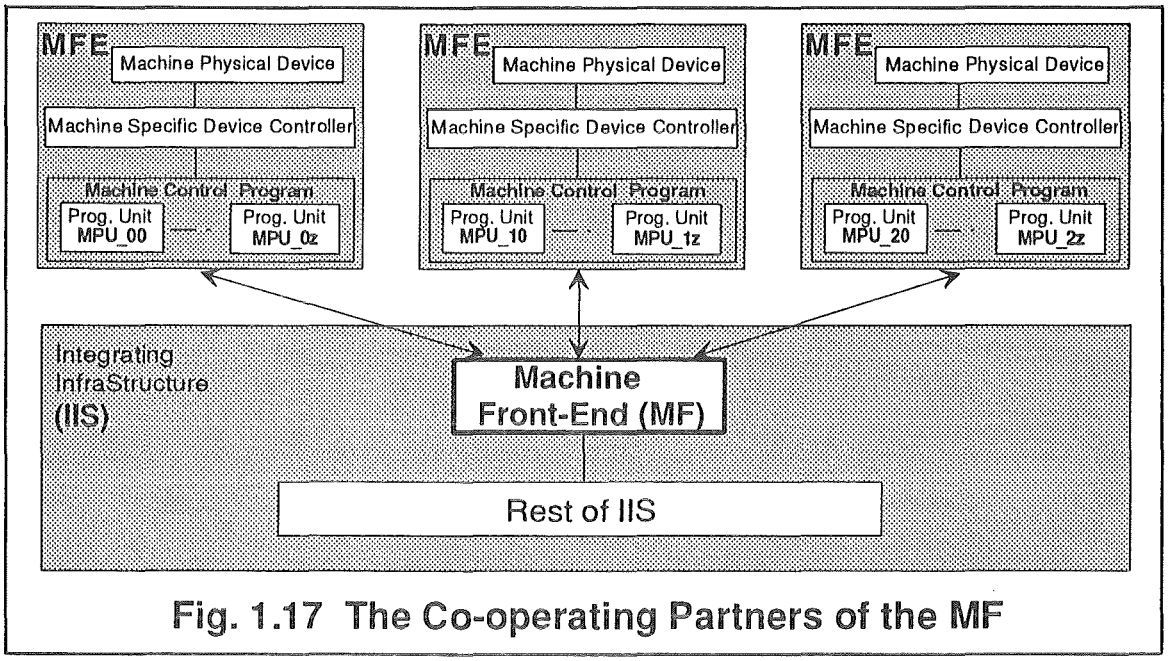


Fig. 1.17 The Co-operating Partners of the MF

1.3.3) State-of-the-Art in the CIM-OSA Development

The goal of CIM-OSA is to upgrade manufacturing enterprises to perform their businesses in a real time adaptive mode. CIM-OSA provides a global solution, by allowing the building of flexible and consistent enterprise models that are designed and maintained using appropriate engineering tools, and are directly executable through an *Integrating Infrastructure* in an operational environment. It improves the enterprise operational flexibility and multi-disciplinary information (knowledge) integration and system integration.

Coming to the conclusion, CIM-OSA provides a *Modelling Framework* to ease the description of the real world of manufacturing enterprises. The modelling

methodology is composed of an integrated set of reference models (generic building blocks) supporting modelling and optimizing the enterprise requirements, design and implementation in terms of functions, information, resources and organization. The top-down approach supports a good means for the overall optimization of enterprise operations. Moreover, it allows the CIM-user to apply user-specific algorithms for the scheduling of enterprise resources. In order to achive the scheduling, the *Resource Mgmt.* of the Business Services is provided to manage the enterprise resources.

The results of a modelling task are executed by the *Integrating Infrastructure* which deals with communication, information, front-end and business process management services to integrate heterogeneous manufacturing and information technology environments.

In comparison with the current software development of a CIM-application shown in Figure 1.6, CIM-OSA provides a complete information process from the modelling of enterprise activities to their operation throughout an enterprise. In addition to the concepts described before, two other aspects are discussed below:

- Separation of the ***Function Initiator*** from the ***Function Executor***:
  As stated before, the current CIM-application involves a number of *functions* so that it acts on the one hand as the *function initiator* and on the other hand as the *function executor.* CIM-OSA separates these two elements. It supports the Business Services acting as the *function initiator* to issue the *elementary Function Models* (i.e FO's) for the execution. The Business Services takes over the task control of the execution of a domain process and requests the Front-End Services to execute FO's. The Front-End Services acts as the *function executor.* In response to this request, the Front-End Services co-operates with external CIM-Modules to achieve the job specified by the FO. This concept improves flexibility for the design and implementation of CIM-OSA models. Therefore, it covers the user needs for the adaptation of their manufacturing processes to continuously changing markets and technologies.
- System Openness and the integration of heterogeneous enterprise resources:
  CIM-OSA offers an architecture which satisfies the requirements for an open system. An open system is characterized by the independence of the system from vendor-specific applications. It enables the CIM-user to concentrate on the description of the functions he needs, he will then have a wide spectrum to select a system from any vendor which can fulfil these functions. However, the openness of a system can only be achieved if there are international standard protocols

applied by vendors and users. The standardization efforts of CIM-OSA are needed not only in the Modelling Framework and the Integrating Infrastructure, but also in the interfaces of Integrating Infrastructure to the external CIM-Modules.

The CIM-OSA concepts of the integration of the modelling task and the execution, the system openess, as well as the availability of standard software modules will provide a great flexibility for enterprises to respond quickly to a changing environment. Therefore, CIM-OSA has been recognized as the most comprehensive and well-structured CIM-architecture for the manufacturing industry [*Remb90a*].

Although, a large number of research institutions, universities and industrial companies have made their contributions to the CIM-OSA development, CIM-OSA is still in a stage of conceptual and global description. At present, the AMICE project is in a consolidation phase. The development of a CIM-OSA IIS prototype, being the main goal of the current AMICE project, was not achieved and so has been postponed for a future project. The CIM-OSA results are evaluated and summarized as follows:

- There are only some generic building blocks defined for the modelling of the *Function View* and *Information View* of the Requirement Definition Level;
- CIM-OSA provides guidelines for building IEE-tools (Integrated Enterprise Engineering) to assist the modeller in handling the building blocks. A prototype of IEE-tool is being developed within the AMICE Project;
- The concepts of the CIM-OSA Modelling Framework, Modelling Building Blocks, IIS Framework and IIS Service/Protocol Definition are foreseen to be internationally standardized. The Framework for Modelling was issued by CEN/CENELEC as the European Pre-Norm ENV 40003, but the current AMICE project concentrates more on the development of the Modelling Building Blocks and the IIS than on the standardization efforts;
- In the IIS there are still some gaps in the definition of services. Only the structure of the Communication Services has been given. Prototyping however needs more detailed specification. The Information Services is the most completely defined service block of the Integrating Infrastructure. A prototype can be made if the interface to the existing relational database management systems is clarified. For the Business Services, the basic ideas have been sketched and some services are identified. However, any implementation needs still a lot of effort for the specification of services in detail and the definition of the internal representation

form of the *Domain Process Model* including the *Procedural Rules*. The Front-End Services contains three types of services which are still black boxes, only their objectives, functionalities are globally described.

This dissertation deals with the development of the Machine Front-End. The Machine Front-End should, on one side, take the control and management of the MFO execution, and on the other side, integrate heterogeneous manufacturing devices. Figure 1.18 is to clarify the relationship between the Machine Functional Operations (MFO's) and the Machine Front-End. The functionality of a MFO is achieved by the co-operation between the Machine Front-End and the Machine Functional Entity (i.e. external machine controller). Therefore, the description of a MFO resulting from the *Modelling Framework* should be implemented in two parts: one is the *control part* in the Machine Front-End and the other is the *execution part* in the external machine controllers. The execution parts are realized by the *Program Units* described before.

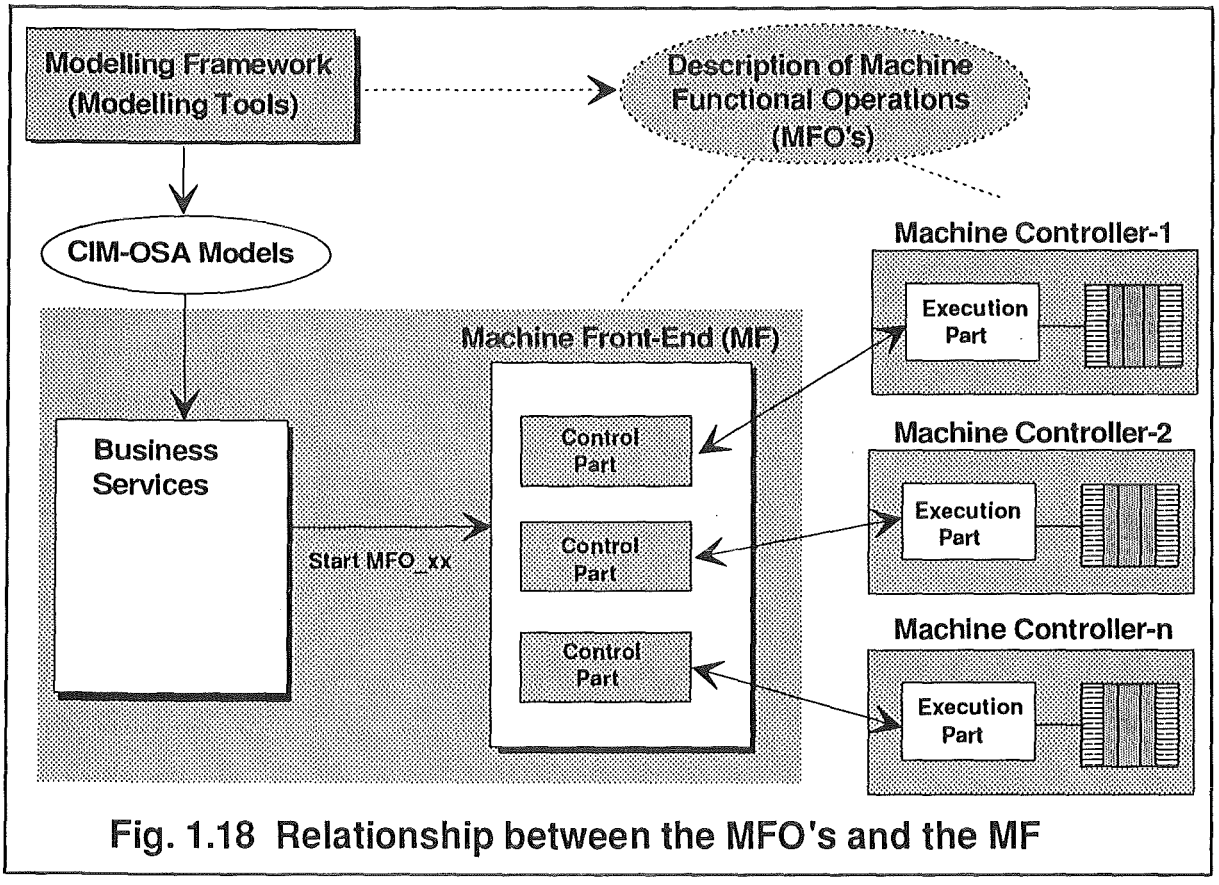


Fig. 1.18 Relationship between the MFO's and the MF

Furthermore, the Machine Front-End has a two-fold nature in the IIS environment of the client-server model: it acts as a client and as a server. The following chapters will describe the problem statement and the proposed solution in detail.

2) Problem Statement and Motivation

This chapter states the problems which arose during the validation task of CIM-OSA. It identifies the objectives of the Machine Front-End development including the definition of requirements and capabilities.

2.1) Problem Statement

After a careful study and analysis of CIM-OSA, it was found that CIM-OSA provides a very promising integrated methodolgy in the area of CIM-models. It is also known that under the *Modelling Framework* as well as the *Integrating Infrastructure (IIS)* there are still many concepts to be validated and explored in further detail.

AMICE provides only a document which consists of various pieces of information about the Front-End-Services [*CIMO90/C5-3000*]. Each piece of information is described in a style of so-called *Formal Reference Base*. From the analysis of this document we recognize that [*GHLS91*]

- the development of Front-End-Services by AMICE is only in the conceptual phase. So far only the objectives, functionalities, some relationships with the external CIM-Modules are described. The content of all three types of Front-End-Services is still a black box;

- there are no products nor specifications available which could be used to implement the concept of Front-End-Services.

Therefore, in the VOICE project the Front-End-Services could be validated with respect to the basic concept only. It was found, however, that the approach of CIM-OSA IIS concept is sound and feasible and therefore worth a prototype-oriented validation. As shown in Figures 1.4 and 1.5, the development of an IIS prototype is also the main objective of the present AMICE project 5288.

From the previous description of all three elements of Front-End-Services, we can find that they are similar in nature:

- They should provide other IIS components with several types of services;

- They should manage and control the execution of *CIM-OSA elementary Function Models* (i.e. *Functional Operations - FO's*);

- They should integrate heterogeneous enterprise resources (external CIM-Modules) by use of External Protocols;

- The external CIM-Modules should consist of a number of *Program Units*. Each of them represents the execution part of a *Functional Operation* resulting from the CIM-OSA modelling framework. These Program Units should perform the protocol conversions to the device-specific protocol and control the actions of physical manufacturing devices;

- The external CIM-Modules are viewed as enterprise resources described by constructs of CIM-OSA *Resource View*.

From the experience obtained during the building of a migration platform [*Hou 91*], the author found that the Machine Front-End has various functions which are similar to the ***Transformation Module*** shown in Figure 2.1. In the CIM-OSA terminology, the Server Station in the figure can be viewed as a Machine Functional Entity (i.e. external CIM-Module). The proprietary distributed application processes act as the Business-Services which issues elemntary functions to the Transformation Module for their execution. However, there are two main problems if we apply this approach to the MF design:

- It may not be possible to represent a Program Unit installed in Machine Functional Entity by just one indication function, i.e. a Machine Functional Operation may need a number of indication functions to accomplish its functionality;

- The capability of the Machine Front-End can only be changed via the configuration of its parameters by an external tool. This means that the modification of program source codes of the Machine Front-End by adding or removing a program unit must be avoided.

Taking all the above constraints into account, the author decided to begin with the MF design and to develop a prototype for the demonstration of CIM-OSA concepts. The development should be based on the CIM-OSA concept and reflects the industrial user requirements. The prototype should become a major basis for the product oriented developments of the Integrating Infrastructure in the future.

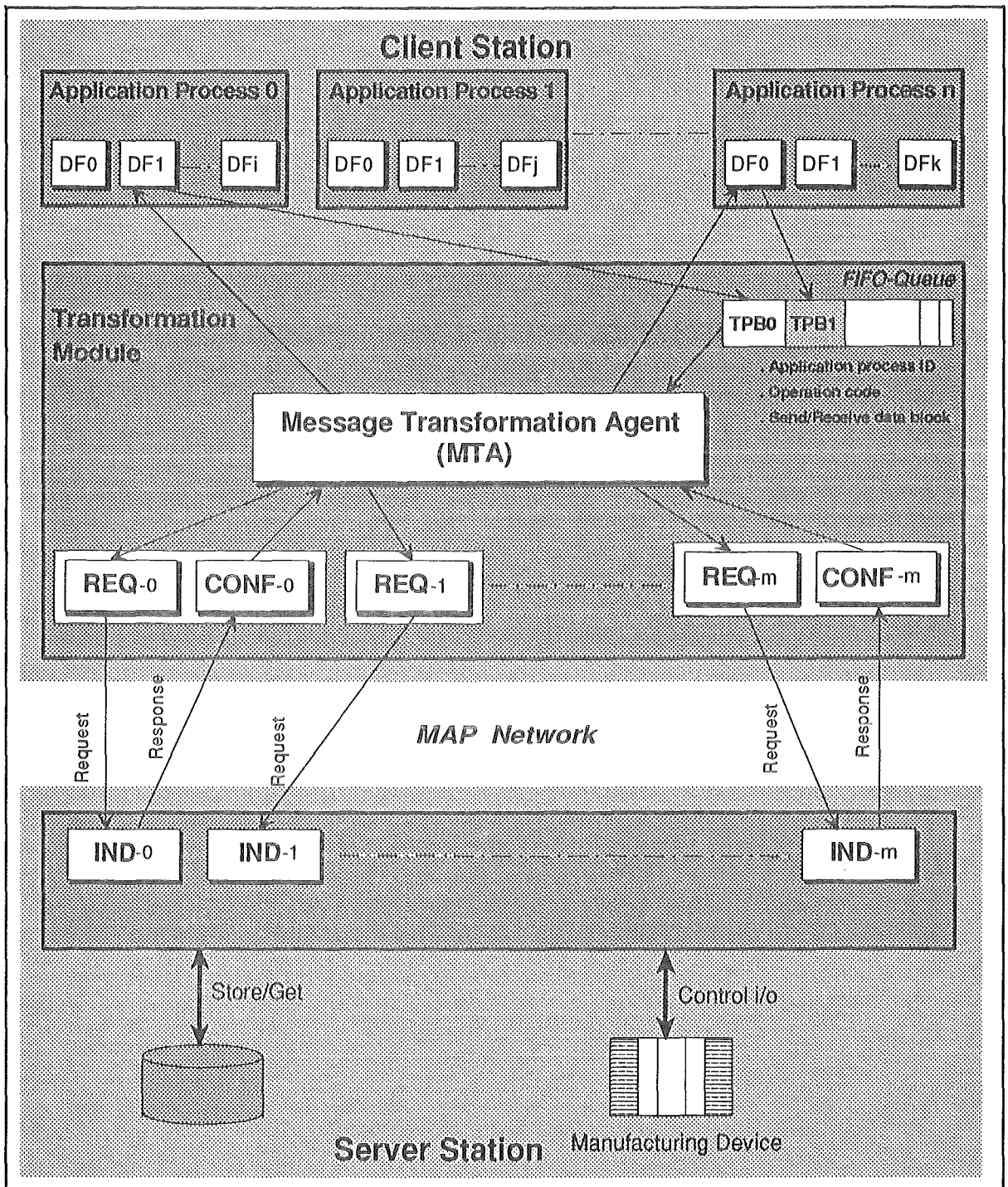


Fig. 2.1 Migration of Proprietary Distributed Application into MAP/MMS

Keys: DFx:proprietary Distributed Function REQx: Request Function CONFx: Confirmation Function
INDx: Indication Function TPB:Transformation Parameter Block

So far, we have learned from the CIM-OSA documents that the description of the *Machine Front-End (MF)* is restricted to the objectives, functionalities and some relationships to the external applications. A guideline for the definition of a set of services for the operational control is also available, but how the control part of an *elementary Function Model* (i.e. *Machine Functional Operation*) is implemented in the Machine Front-End and how the Machine Front-End co-operates with other IIS components as well as with the machine controllers is not clear. In other words, there is no methodology described by AMICE for the MF design and also no approach available in the research area which can be used to implement the Machine Front-End.

2.2) Objectives of the Work

This work will emphasize on the design of a methodology for the Machine Front-End and thereafter a prototype will be developed as a basis for the CIM-OSA demonstration. The development of the Machine Front-End should on the one hand conform to CIM-OSA concepts and OSI standards, and on the other hand satisfy the industrial user requirements.

The conformance of the MF development to CIM-OSA concepts means that the Machine Front-End should support the execution of CIM-OSA models in cooperation with other IIS components, particularly the Business Services. It should be able to integrate heterogeneous manufacturing devices. The following concepts of the Machine Front-End must be realized for the MF development:

- In the IIS framework, the Machine Front-End should act on one side as a server to its clients (MF-Clients are Business Services, Human Front-End, etc.) and on the other side as a client to its servers (MF-Servers are e.g. machine controllers).

- According to the CIM-OSA Modelling Framework, a Machine Functional Operation (MFO) is defined as the smallest unit for accomplishing a job specified in the CIM-OSA model. However, the functionality of a MFO can not usually be implemented by use of a single underlying MMS service primitive. In order to provide a freedom of description and implementation of MFO's, and to ease the migration of proprietary machine controllers into the CIM-OSA system, the Machine Front-End should have some knowledge within it for taking over the control and monitoring of the MFO's execution.

- Communication with shop floor devices should be preferably based on the functional profile of the international standard *MAP* (*Manufacturing Automation Protocol*) [*MAP89, COMP89, CONC90b*]. The specific model of shop floor devices and its communication protocol must be transparent to the MF-Client.

Furthermore, the Machine Front-End should be able to deal, sequentially or in parallel, with a number of Functional Operations requested by several MF-Clients. It should be able to control and manage several manufacturing devices concurrently.

Satisfication of the industrial user requirements is also an important factor for the MF development. From the analysis of the VOICE industrial partners and some reports of the AMICE projects, the most important user requirements to the Machine Front-End have been summarized. These user requirements are ***vague*** in their description and not precise. In practice, it is often difficult to define precise requirements because of the many scenarios possible in any given application, therefore the user requirements given below are generalized to encompass all possible eventualities:

a) remote monitoring of the production processes, shop floor devices;
   In an assembly or production line, an operator often needs to know the status of the concerned production processes or shop floor devices. The status data captured for the monitoring needs to be displayed immediately to the operator.

b) automatic production data collection;
   There is a quite lot of data in a manufacturing enterprise which is needed later for analysis. This kind of data is captured and stored in a data base first.

c) control, synchronization of shop floor devices;
   The control and the synchronization of activities of several shop floor devices are basic requirements in manufacturing processes.

d) alarm signal and error condition handling;
   When a status data is beyond the allowed range, the system should alarm the operator to readjust the system to the normal range.

e) emergency condition handling, e.g. a stop action by an operator;
   The emergency signal indicates that the system state is critical and could damage the system. Therefore, the system should support the operator to immediately take actions, e.g to stop the process.

f) time critical operations;

In manufacturing processes, some operations are required to be accomplished in a given time interval. The time interval required is dependent on the process, which can range from miliseconds to several minutes.

g) local and remote access to MMS Servers (Manufacturing Message Specification) and non-MMS Servers;
The access and location of diverse manufacturing devices should be hiden from the system user and should be easily configurable.

h) Integration of multi-vendor devices;
This deals with facility for the integration of heterogeneous devices.

i) support for the design and implementation of CIM-OSA models;

j) user oriented integrated engineering tools for the configuration of the CIM-OSA system;

The requirements: (a) - (h) represent the additional capabilities which the Machine Front-End should possess. The items (i) and (j) reflect the engineering support for the building of CIM-OSA models, they belong to the scope of the CIM-OSA Integrated Engineering Tools (IEE-Tools). From the CIM-OSA concept, the two user requirements, time-critical operation and emergency handling, are discussed below.

The Integrating Infrastructure (IIS) acts as a cell controller in a manufacturing cell and needs several processes to achieve its functions. The IIS processes should be installed on a multi-tasking operating system which will influence the execution time of a CIM-OSA model in addition to other factors such as the underlying network communication subsystem, machine control system, etc. Therefore, the Integrating Infrastructure is used more for the overall control than for time-critical operations. The operations to be accomplished within the miliseconds range should be put down to the machine station level. However, some mechanisms, such as a priority-driven execution which allows the process of the models with higher priority first, can be introduced in the MF design.

Both the 'error condition handling' and the 'emergency handling' may need intervention by an operator, this means that the Human Front-End must provide the facility to mediate with an operator. Since the three elements of the Front-End Services have similar features, the developed MF prototype can be also used to co-operate with an operator. However, this work does not intend to define any External Protocols for the Human Front-End or for the Application Front-End.

3) Conceptual Basis of the Machine Front-End (MF) Design

The chapter describes the conceptual basis applied to the MF design. Beginning at the discussion on the features of the Machine Front-End, an advanced approach for the MF design is presented. The approach consists in providing a Control Model Libray and a Control Engine and introduces the concept of Service Units.

3.1) Features of the Machine Front-End

In the Client-Server Architecture of the *Integrating Infrastructure* (*IIS*), the Machine Front-End has three features:

- a twofold function, as a client and as a server;
- a provision of specific services for the other IIS components;
- an application program for the CIM-OSA users, not a service provider.

A twofold function as a client and as a server

The message passing between two co-operating partners in the CIM-OSA IIS environment is based on the ***Client-Server Architecture***. According to the MMS-Service definition, the ***Server*** is defined as the peer communicating entity which behaves as a *VMD* (*Virtual Manufacturing Device*) for a particular service request instance. The ***Client*** is the peer communicating entity which makes use of the VMD for some particular purpose via a service request instance [*ISO 90*].

The Machine Front-End is not simply a server. It acts both as a server to its clients (Activity Control, Resource Mgmt. Human Front-End, etc.) and as a client to its servers (e.g. machine controllers). The client-server relationships of the Machine Front-End to its client and server are described in Figure 3.1. According to a MF-Service Request (MFO call) sent by MF-Client, an appropriate MF-Service Indication Function is started. Within this function a number of service requests with associated request data are formed and then sent to the MF-Server. The service requests issued by the Machine Front-End have other service classes which are based on the agreement between the Machine Front-End and its server. They should be understood by the MF-Server in order that it can take the desired operations.

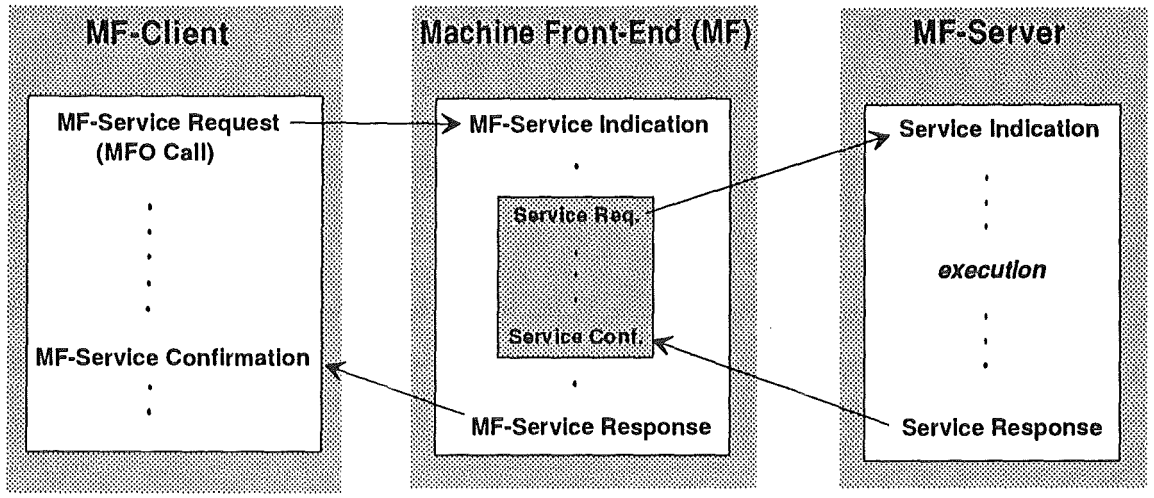


Fig. 3.1 Client-Server Relationships in the MF Environment

The Machine Front-End acts as an agent between the service requestor (MF-Client) and the service executor (MF-Server). After receiving a MFO call, it generates corresponding service request(s) and sends them to the appropriate MF-Server.

In the CIM-OSA IIS environment, the Machine Front-End should be able to deal, sequentially or in parallel, with multiple *MF-Service Requests* (*MFO calls*) from more than one MF-Clients.

Figure 3.2 shows the relationships of the Machine Front-End with two clients and two servers. It describes in detail the functional interactions of the Machine Front-End with its clients and servers. Each block represents a program function. *MF_xxx_req* indicates a callable function supported by the Machine Front-End and *ss_xxx_req* is labelled for a callable function supported by MF-Server (a machine control system or any other IIS component). The character *u* indicates that the function is a user-specific function, and the xxx indicates a specific service type. Within the execution of a *MF-Indication Function* (*u_MF_xxx_Ind(...)*) several service requests may be sent to the MF-Server to ask for the execution of specified jobs. After the Machine Front-End has sent a service request (*ss_xxx_req(...)*) of a *MF-Indication Function*, it should be able to continue to handle the other *MF-Indication Functions* without waiting for the response data.

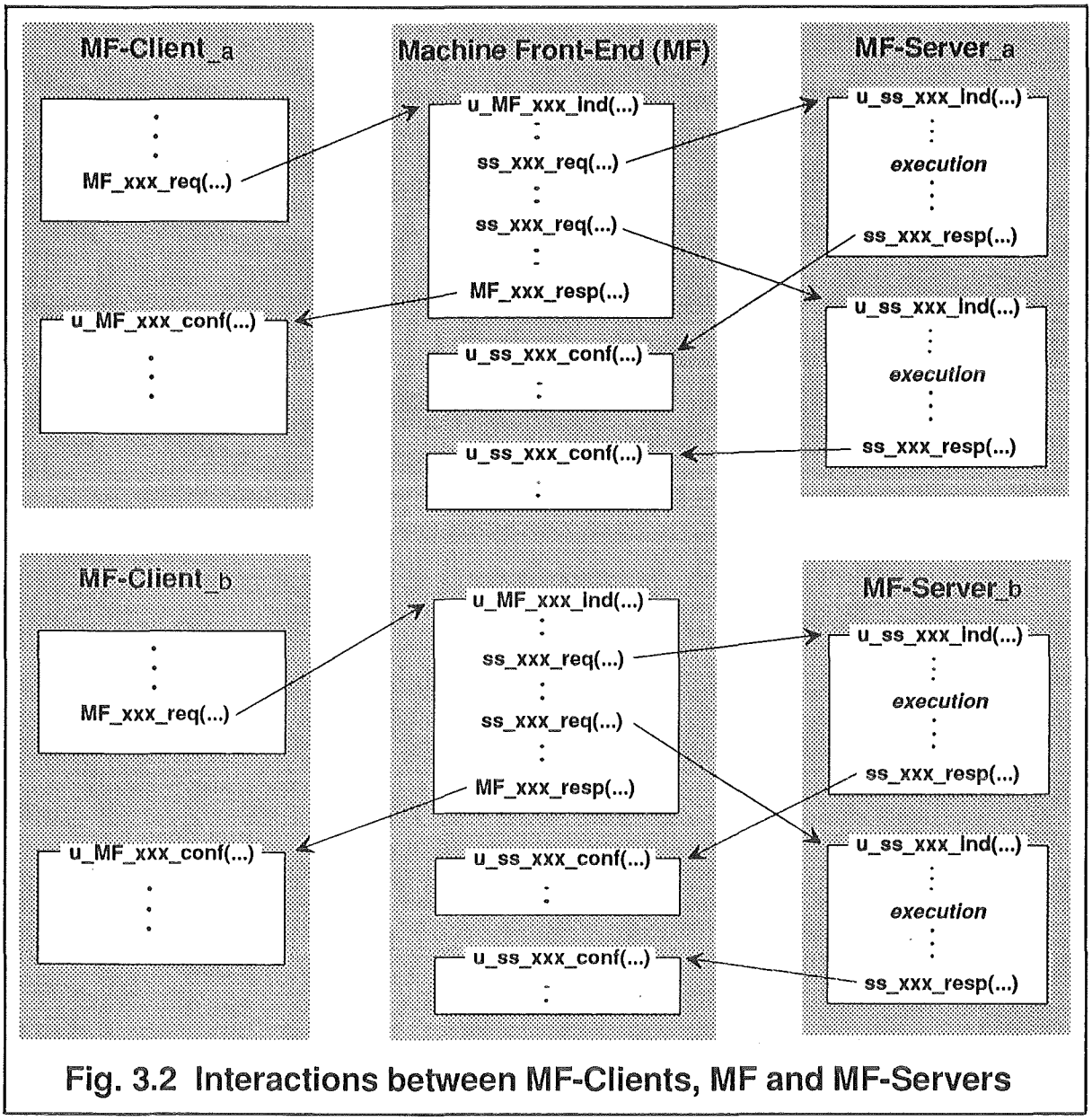


Fig. 3.2 Interactions between MF-Clients, MF and MF-Servers

A provision of specific services for the other IIS components

For the interaction between the Machine Front-End and its clients, the Machine Front-End should provide several sets of services to its clients.

An application program for the CIM-OSA users, not a service provider

A *service provider* supports a set of services which can be called by a main program, so they can be linked together as a task. A service provider can also be implemented as a task which provides other tasks with services via data exchange. The Machine Front-End is a service provider to the Business Services, but it is not a service

provider to the CIM-OSA users (process modeller, implementor of CIM-OSA models, and operator) because it (actually all the IIS processes) provides no services to the CIM-OSA users. It is an *application program* which is transparent to the CIM-OSA users who don't need to know about the Machine Front-End.

3.2) A Possible Solution to the MF Design and the Problems

One straightforward solution to the MF design might consist of building a necessary set of indication/confirmation functions within the Machine Front-End. Based on this solution the Machine Front-End can be designed as shown in Figure 3.3.

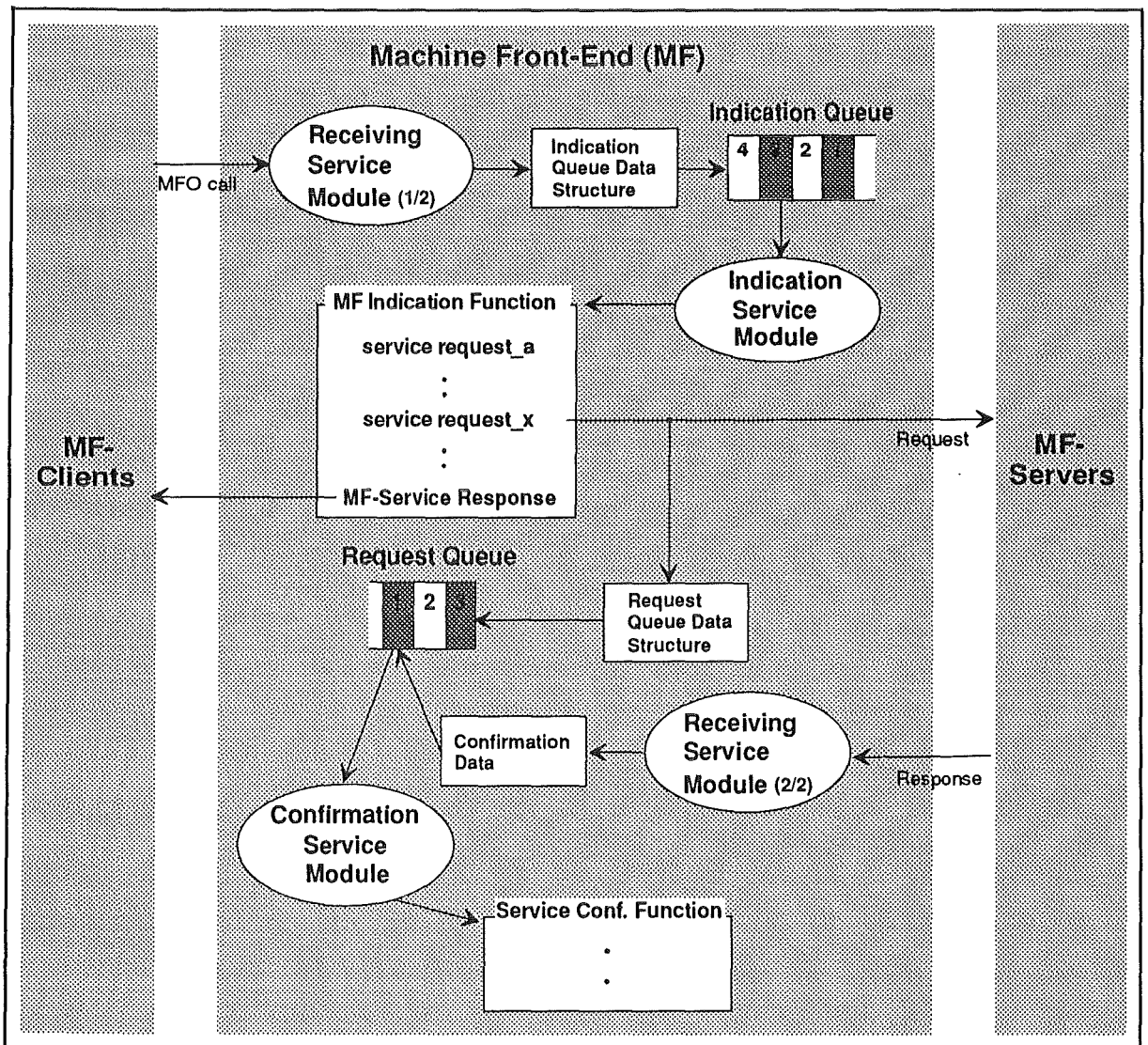


Fig. 3.3 The MF Structural Components

The Machine Front-End processes two kinds of input messages and two kinds of output messages. The input messages are the *MF-Service Requests* (*MFO calls*) from MF-Clients and the confirmation messages from MF-Servers. The output messages are the service requests to MF-Servers and the *MF-Service Responses* to MF-Clients. For tracking each of the outstanding messages two queues are applied, the *Indication Pending Queue* and *Request Pending Queue*. The *Indication Pending Queue* is used to pipeline the outstanding MF-Service Indications, and the *Request Pending Queue* is used for the outstanding service requests.

When a *MF-Service Request* (*MFO call*) is received, the Machine Front-End parses the Protocol Data Unit (PDU), then assigns an indication queue data structure to it. This indication queue data structure will then be pipelined into the *Indication Pending Queue*. In consequence of this, the *Indication Service Module* will pick up an outstanding indication from the Queue according to the FIFO (First-In First-Out) principle and will call the appropriate *MF Indication Function*.

Similarly, when a service request is issued, the Machine Front-End assigns a request queue data structure to it and pipelines this data structure into the *Request Pending Queue* for tracking the outstanding requests. As soon as the Machine Front-End receives confirmation data from its server, it will examine the outstanding requests to pick up the one for which its confirmation data was meant. The confirmation data is stored together with a pointer to the corresponding request. The *Confirmation Service Module* is therefore called to process the corresponding *Service Confirmation Functions*. The resulting data may be needed later to form the next request data or the MF-Service Response data, which however is not shown in the figure.

At the end of execution of a *MF-Service Indication*, the MF will send response data back to the MF-Client and then remove all the associated queue data for the *MF-Service Indication* (requests, confirmations and indication) from the two queues.

Figure 3.3 also shows the entire operating procedures of the Machine Front-End described above. Three service modules are applied to process the queue data and pass the control to the *MF-Indication Functions* and the *Service Confirmation Functions*. These are *Receiving Service Module*, *Indication Service Module* and *Confirmation Service Module*. They will be periodically called by the MF.

However, in this kind of solution, some disadvantages can be anticipated:

- Recompilation of the MF source program:

  Even for a small change in a *MF-Service Request* (*MFO call*), the service requests which will be sent to the MF-Server may need an appropriate modification. This in turn will require the recompilation of the MF source program. In order to avoid this recompilation, a large library of indication/confirmation functions for each type of *Machine Functional Operation* must exist. The problem is, that even a large program does not guarantee that the appropriate indication/confirmation functions that match the requirements of the new *Machine Functional Operation* will be found in this library.

- Inability to handle the time critical MFO execution:

  An interruption of an indication function which is being processed needs more sophisticated design. The indication functions can only be processed by the Machine Front-End one after another. Even the request of a time critical MFO execution has to wait until the last processing function is completed. This also causes the MF to spend more time in a wait-state, i.e. the efficiency of the MF is greatly reduced.

- Problems with the implementation and portability of Function Models:

  Each Function Model is implemented in two parts: the execution part as a program unit in the machine controller and the control part as an indication function in the Machine Front-End. The co-operation of these two parts achieves the functions of the represented *Machine Functional Operation*. Adding a new or removing an existing indication function to/from the MF source program may cause additional problems because indication functions are part of the MF source program. It is also difficult to carry these two parts to other machine controllers for the same Function Model, i.e. one of the CIM-OSA goals, the provision of standard software modules, can thereby not be reached.

- Complication in the management of MF Abstract Objects:

  The services provided in a client-server model are based on the Abstract Objects defined for the server (see Chapter 6). The services acting on an Abstract Object should be designed as being complementary between client and server. In the IIS environment the Machine Front-End should be able to co-operate with several IIS components. This means that the Machine Front-End may have a number of

Abstract Objects and these objects must be managed in a way that multiple service requests from the MF-Clients can be served efficiently and concurrently. However, the management of the MF Abstract Objects in such an approach of the MF Design needs information about the connection of user indication/confirmation functions with the Abstract Objects. This will make the implementation more complicated.

3.3) Basic Concept of the Approach to the MF Design

An advanced approach is introduced in this section, which is:

- to reach the two objectives of the MF design described in Section 2.2;
- to meet the three MF features given in Section 3.1;
- to overcome the disadvantages caused by the solution discussed in Section 3.2.

As stated, every indication function contains the control knowledge of its related *Machine Functional Operation (MFO)*. The content of an indication function depends on the design specification of MFO's resulting from the CIM-OSA modelling. Therefore, the control knowledge contained in the indication functions is MFO-specific (Function Model-specific).

The Machine Front-End is not defined as a ***service provider*** such as MMS and FTAM. It is an ***application program*** for the CIM-OSA users (process modeller, implementor of CIM-OSA models, and operator). Even the implementor of the CIM-OSA elementary Function Models (i.e. MFO's) doesn't need to know about the Machine Front-End.

Therefore, the indication/confirmation functions should be extracted from the Machine Front-End, i.e. the MFO-specific control knowledge should be separated from the generic control mechanism of the Machine Front-End. The separation of these basic elements has crucial importance for the MF development. This allows the problems to be solved, which were stated in the last section, and supports the openness of the development.

Following this basic strategy, a structure of the Machine Front-End is given in Figure 3.4 with the relations to its multiple clients and servers. The Machine Front-End consists of a ***Control Model Libray*** and a ***Control Engine***.

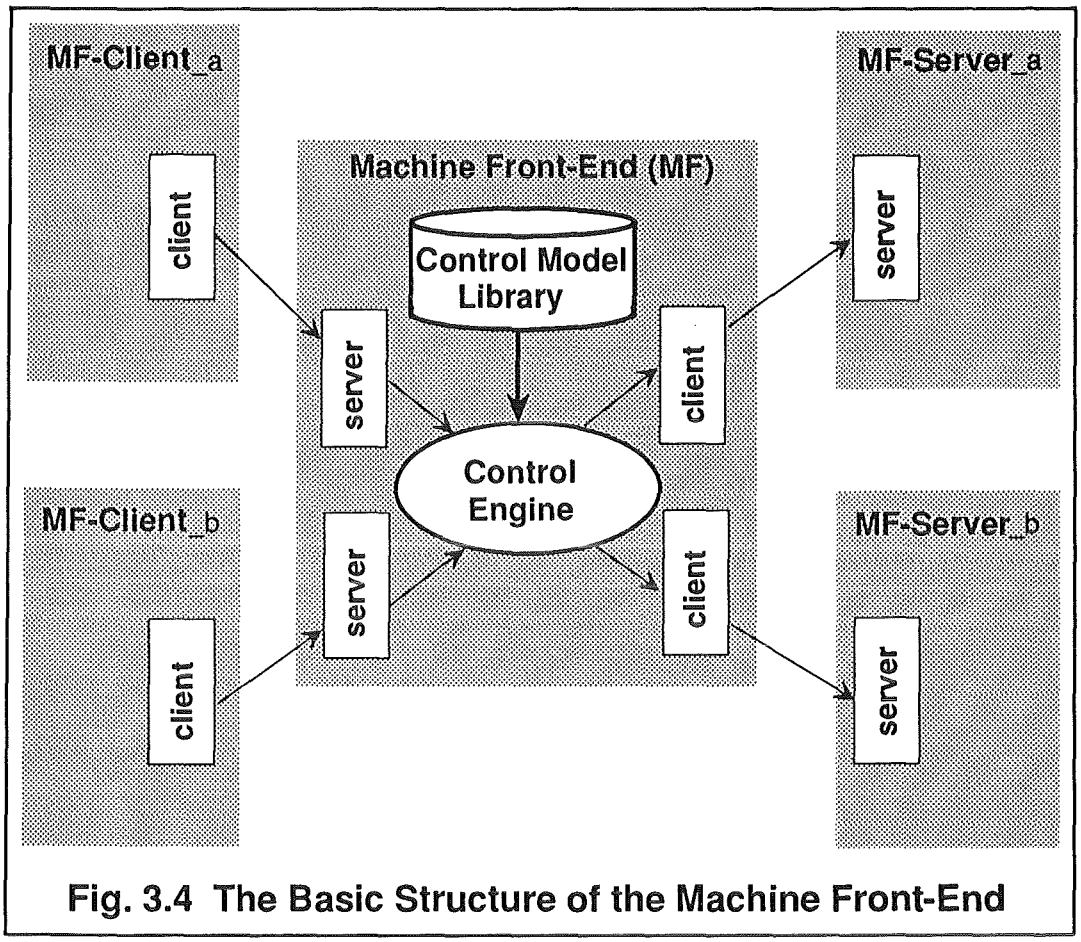


Fig. 3.4 The Basic Structure of the Machine Front-End

The ***Control Model Libray*** actually represents a database containing control models (all the MFO-specific control knowledge of an application) to be interpreted by the Control Engine. It participates in the transformation of the *MF-Service Indications* (*MFO calls*) into a sequence of service requests to be sent to its MF-Servers.

The ***Control Engine*** is the kernel program of the Machine Front-End. It contains the generic control mechanism which manages the pending indications and service requests, selects the appropriate control knowledge and controls the MF Abstract Objects, etc. It includes a *Transformation Agent* which uses the *Control Model Libary* to control and monitor the MFO execution at the MF-Servers, instead of calling the user indication/response function.

Figure 3.5 gives details on the structural components of the Machine Front-End. It is designed according to the proposed approach.

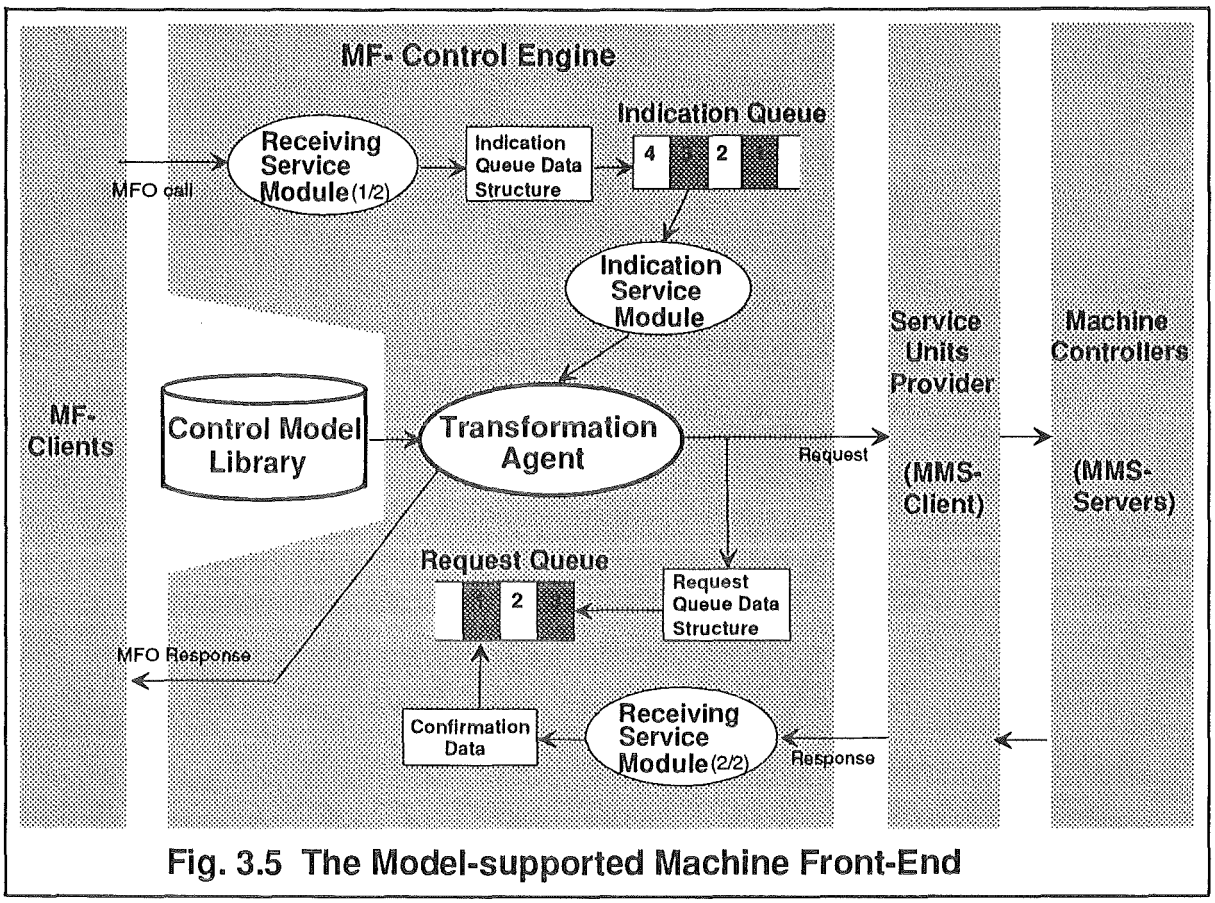


Fig. 3.5 The Model-supported Machine Front-End

The ***Control Model Library*** contains the MFO-specific control knowledge. As can be seen, the *MF-Indication Functions* and *Service Confirmation Functions*, shown in Figure 3.3, are replaced by the *Control Model Library*. The ***Control Engine*** implies three service modules: *Receiving Service Module*, *Indication Service Module* and *Transformation Agent*. It arranges the incoming messages in two queues (*Indication Pending Queue* and *Request Pending Queue*), manages the *MF Abstract Objects* and processes the *Control Model Library*. The *Confirmation Service Module* shown in Figure 3.3 is included in the *Transformation Agent*.

The *Receiving Service Module* is an interface for the message passing. It is responsible for receiving and checking the messages for the Machine Front-End. In the framework of the Integrating Infrastructure it will be implemented by use of the callable functions of the System Wide Exchange. However, other facilities for message passing can be applied in this module as well. This enables the Machine Front-End to be more adaptive to its environment.

The *Indication Service Module* reads an indication message from the *Indication Pending Queue* placed there by *Receiving Services Module*. It generates an appropriate control record for the *Transformation Agent* to control the execution of the specified Machine Functional Operation.

The *Transformation Agent* is the central part of the *Control Engine*. It transforms the *MF-Service Indications* (*MFO Calls*) received from clients into a sequence of service requests for sending to its servers (e.g. MMS Devices) by use of the *Control Model Library*. It registers the issued request data and manages the received confirmation data for the use of the succeeding requests to be issued.

Similar as in Figure 3.3, two queues, the *Indication Pending Queue* and *Request Pending Queue*, are applied to track the outstanding messages. They contribute also to the capabilities of the Machine Front-End with respect to processing multiple indications and requests concurrently.

In this approach of MF design, except the basic strategy discussed above which separates the application-specific control knowledge of Function Models from the generic control mechanism, the Machine Front-End uses the concept of *Service Units* and applies the *object modelling technique* to specify the capability of the Machine Front-End. These will be discussed in detail in the following chapters.

4) The Control Model Library

The *Control Model Library* is the first fundamental part of the proposed Machine Fron-End. It contains the application-specific control knowledge of *elementary Function Models*(i.e. *MFO's*), and consists of a number of Control Models. Each Control Model is responsible for the control of execution of a specified MFO. Several advanced techniques, such as AI- or object-oriented techniques, have been investigated for the model representation. They appeared, however, as being not suitable in the context of the MF design and finally a command language was chosen. This chapter describes the structure of the Control Model Library and some important key elements of the command language used for the building of the Control Models. The Abstract Objects are discussed in Chapter 6.

4.1) Information Tree of the Control Model Library

The *Control Model Library* contains the application-specific control knowledge of Function Models resulting from CIM-OSA modelling. It consists of a number of ***Control Models*** such that each of them participates in transformation of *MF-Service Request* (call to a specified *Machine Functional Operation*) into sequences of service requests for sending to the MF-Servers.

In the CIM-OSA IIS environment, data communication with the Machine Front-End (MF) can be divided into two types: the communication with other IIS components and communication with external machine controllers. The first type can directly use the services provided by other IIS components. For the second type two considerations should be taken into account:

- To accomplish a MFO execution one may need a number of interactions between the MF and the MFE's (i.e. machine controllers). As stated before, the functionality of a MFO can not be usually mapped exactly to, and accomplished by a single MMS service primitive. Therefore, for each type of MFO execution a Control Model should be provided and contain the expert knowledge which contributes to the control of interactions between the MF and the machine controllers.
- The MF should be able to handle the diverse types of machine controllers. The protection of the existing investments has been recognized as one of the most important requirements of the CIM-user for building a CIM-OSA system. The

integration or migration of the existing proprietary machine controllers into a CIM-OSA system should be possible and cost effective.

The content of a ***MFO Control Model*** is dependent on the functionality of its represented MFO, which can be just a simple sequential control or a very complex control algorithm. It may also consist of a number of interactions with other entities: e.g. data accesses to a database or filestore system, mediating with human operator, data exchanges with other applications, or interactions with other machines. However, the support for the representation of *MFO Control Models* should cover the user needs. In the section 4.3 several techniques will be discussed.

All MFO Control Models of the library should be organized in a special form such that they can be easily accessed. Figure 4.1 describes the information tree of the *Control Model Libary*. The capability of the Machine Front-End is defined by a number of *Abstract Objects*, which applies the object modelling technique described in Chapter 7. An ***Abstract Object*** is used to define a set of supported services by the Machine Front-End for the MF-Clients. Several *Abstract Objects* of the Machine Front-End are identified by CIM-OSA, which are e.g. for the Operation Control, Resource Mgmt., Information Store, etc. [*CIMO90-C5-3000*]. All the MFO Control Models are linked to their related *Abstract Object* and are the bottom elements of the information tree.

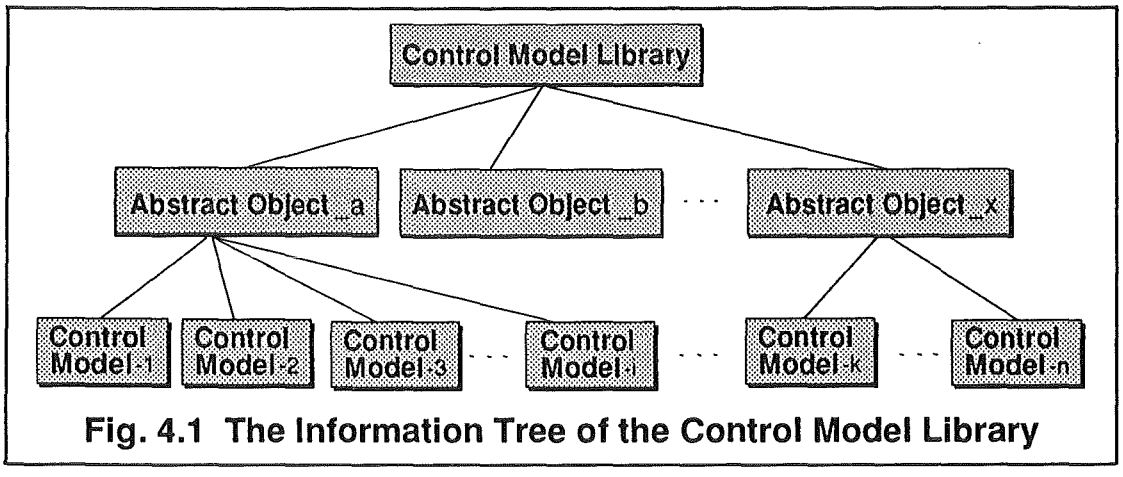


Fig. 4.1 The Information Tree of the Control Model Library

Depending on the received MF-Service type, a MFO Control Model will be triggered by the *Transformation Agent*. As stated before, to achieve the functions of a specified MFO type, a MFO Control Model must co-operate with the Program Unit installed in the machine controllers. So, the MFO Control Model as well as the corresponding Program Unit must be consistent with the design specification of the

MFO resulting from the CIM-OSA modelling. The implementation of Program Units can be influenced by the machine environment, e.g. the services and protocols used, but the Control Models should be kept independent of the machine environment.

4.2) Control Model of the Machine Functional Operation (MFO)

As we have seen before, the functionality of a MFO can only be achieved by the co-operations between the Machine Front-End and the external machine controller. It is implemented in two parts: the control part embedded in the Machine Front-End and the execution part in the external machine controller (cf. Fig. 1.18). In the proposed MF design, the application-specific control parts of Function Models (i.e. MFO's) are implemented in a separate library consisting of a number of Control Models. Each Control Model holds knowledge to control the execution of its corresponding Function Model.

The Integrating Infrastructure of CIM-OSA is placed on top of the OSI-Application Service Elements (e.g. MMS, FTAM). Therefore, the full utilization of the facilities of underlying processes to ease the implementation of the MFO Control Models becomes an interesting aspect. These facilities are e.g. MMS, FTAM, Remote Procedure Call, distributed operating systems. The following advantages will appear obvious if some functions of the control part of a MFO are extracted from the Control Model and embedded in the underlying process:

- It will ease the task on the implementation of the MFO Control Models. A Control Model may contain only a sequence of *Service Unit Identifiers* and some instructions for preparing the input data for those *Service Units*. The *Service Unit Identifiers* will be activated by the underlying process, called *Service Unit Provider*, which consists of a number of *Service Units*. A ***Service Unit*** is defined to achieve a closed and consistent function.
- The developed *Service Units* can be re-used by other MFO Control Models;
- The MFO Control Models will become independent of the protocols used by the underlying process and the remote machine controllers. In other words, the technology dependency has been moved from the IIS down to the Service Unit Provider, which will enable the creation of the Machine Front-End widely applicable for diverse types of machine controllers and also ease the integration of heterogeneous manufacturing devices.

The concept of *Service Units* applied in this approach of the MF design is consistent with the term of External Protocol identified by CIM-OSA, although still none of the External Protocols are declared there.

An example of data flow within a MFO Control Model is presented in Figure 4.2. Any previously received data can be used as the input data to the consecutive Service Units. According to the received response data the Machine Front-End can send a request to other IIS components, such as Human Front-End, to ask for information for the next Service Unit. The MF-Service Response data may consist of some received data units from the MF-Server.

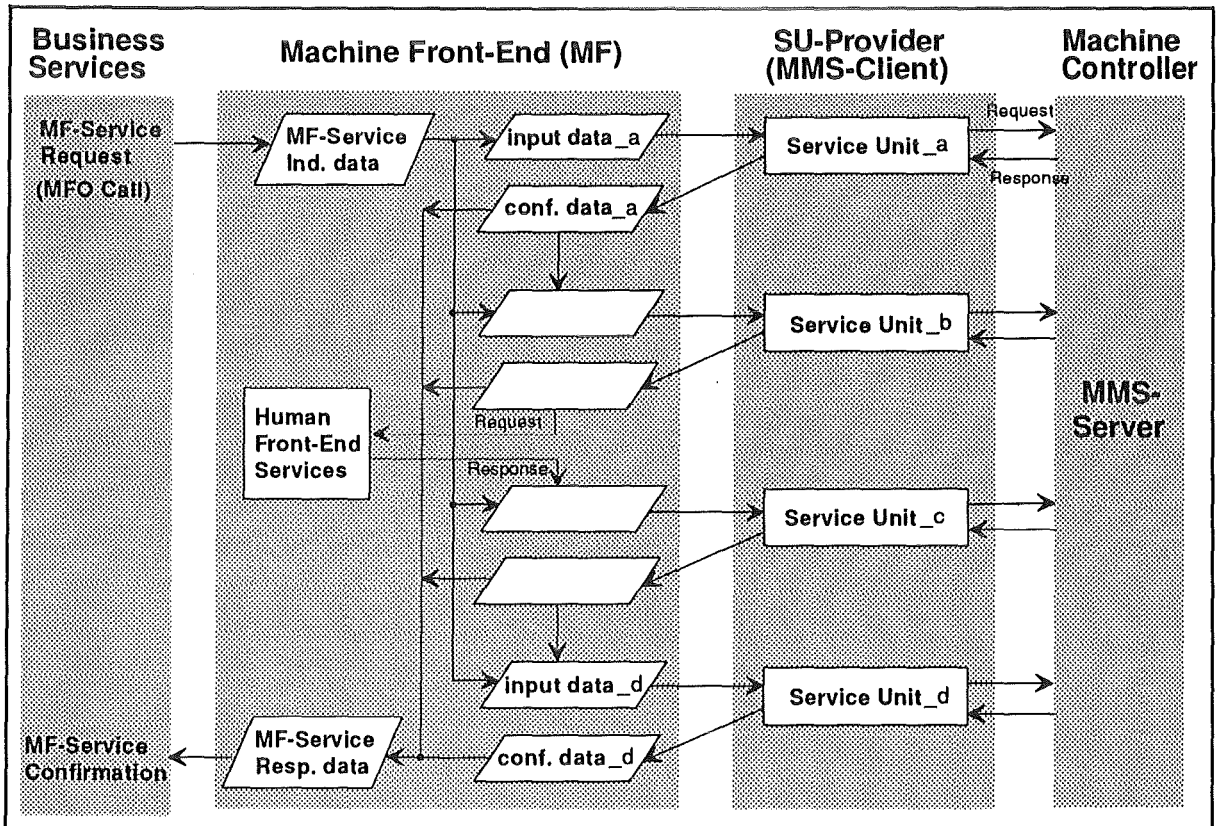


Fig. 4.2 An Example of the Data Flow within a MFO Control Model

4.3) Model Acquisition and Representation

Some questions will arise when the Control Models of the library are going to be put into the computer for processing:

- How to support the application developer to capture their knowledge for building Control Models ?
- How to present the Control Models in a computable way ?
- How to access and process several Control Models in a quasi-parallel way ?
- Which *MF Access Protocols* and *MF External Protocols* should be provided ?

For the capture and the representation of Control Models, it is worth examining the advanced techniques of information technology, such as AI-(Artificial Intelligence) or object-oriented technique.

In the AI, the techniques of knowledge acquisition and knowledge representation are widely applied in knowledge base systems. The knowledge acquisition deals with the process of extracting, structuring, and organizing knowledge from sources, such as textbooks, reports, data base, case studies, empirical data, and particularly human experts, so it can be used in a program [*Atty85*, *Hou 87*]. This is the first task of knowledge engineering in building a knowledge base system. After the acquisition process, the knowledge has to be structured in a way that can be processed by a computer program. The well-known methods for representing the knowledge in a computable way are [*RoWL83*, *Jack86*]:

- Rule-based method: The knowledge is organized as a set of facts and rules. A rule is a formal way of specifying a recommendation, directive, or strategy, expressed as IF *premise* THEN *conclusion* or IF *condition* THEN *action*;
- Frame-based method: The knowledge is organized as a frame hierarchy for inheritance and procedural attachment. A frame is a network of nodes and relations organized in a hierachy, where the topmost nodes represent general concepts and the lower nodes more specific instances of those concepts. A node may stand for an object, concept or event and is described in terms of slots (attributes) and their values. A slot can have procedures (e.g. If-added procedure, If-removed procedure, If-needed procedure, etc.) attached to it. They are executed when the value of the slot is changed. These procedures may then modify values in other slots, continuing the process until the desired goal is achieved;
- Semantic net-based method: The knowledge is organized as a network of nodes, standing for concepts or objects, connected by arcs describing the relationships

between the nodes. This method is often used to represent a subset of a natural language. A semantic net system can be considered to be a frame-based system.

Accompanying with the advanced AI techniques, a number of commercial knowledge base-building tools have already allowed the user to represent the knowledge in a hybrid way, which supports all of the above mentioned methods. The AI-technique allows experts to express their heuristic knowledge in a natural way. Their expertise can be used by other people by means of induction/deduction reasoning techniques. However, because of indeterminate time behaviour in the inference processing, caused by the so-called *Combinatorial Explosion* and *Garbage Collection [WiHo89, Appe85]*, the technique is not very suitable for application of real-time control environment [*Kern89*].

Recently, the *object-oriented approach* has become a recognized method, not only in the research institutions, but also in the industrial applications which are applied in the programming paradigm, database systems, etc. By using the concept of hiearchical classes and instances, the object-oriented design methodology brings mainly two advantages into information technology. These are the capability of inheritance and the re-use of the defined objects and methods [*Pime90, Borl92*].

In the case of MF design, every Control Model is responsible for the control of its corresponding MFO. It is independent of the other Control Models, indeed there is no relationship between these models. Therefore, we can simply apply the technique of a *command language* for the representation of the Control Models. This traditional technique has the advantage of a determinitive performance. It is used particulary in the manufacturing environment, such as the two wide-spread implementations of international standard ***MMS*** (*Manufacturing Message Specification*) [*ISO 90, SISC90*] and ***RPC*** (*Remote Procedural Call*) [*SUN 90*].

Since the model building is the task of the application developer, being viewed as an expert in the field, so the support for acquiring expertise can be limited only to the support of a comfortable user interface. It is well-known that the formation of knowledge representation is very dependent on the kind of knowledge to be applied. It is known that the simpler the forms of the suopported model representation is, the less and the more constant processing time the *Transformation Agent* needs. Consequently, it will reach a higher performance which plays a very important role in the manufacturing processes. This in turn promises the success of this MF-design. The criteria of the support for the model representation are identified as follows:

- The formation for the model representation should, if possible, provide just a simple form, as long as it can cover the industrial user needs. Complex forms which could cause a long and varying processing time should be avoided;
- It should be expandable;
- It must be easy to use by the application developer without additional learning;
- The implemented Control Models must be independent of the services and protocols applied in the machine controllers.

When looking for an appropriate command language, two packages were choosen, the *AEG-GeAmatics Production Control System* [*AEG 90*] and the *EasyMap System* [*PROC90*]. These were discussed and evaluated within the cooperation task between both VOICE and AMICE Projects. However, these were not accepted because of the need for extensive adaptation of these systems to the CIM-OSA environment and the problem on obtaining source codes. Therefore, a decision was made to define a command language which can achieve the concept of the MF design described before. Some important key elements of the language and its use are discussed in Section 4.4.

Figure 4.3 sketches the processing procedure of the Control Model Library from the capture of Control Models to the generation of the operational models which can be accessed by the Control Engine.

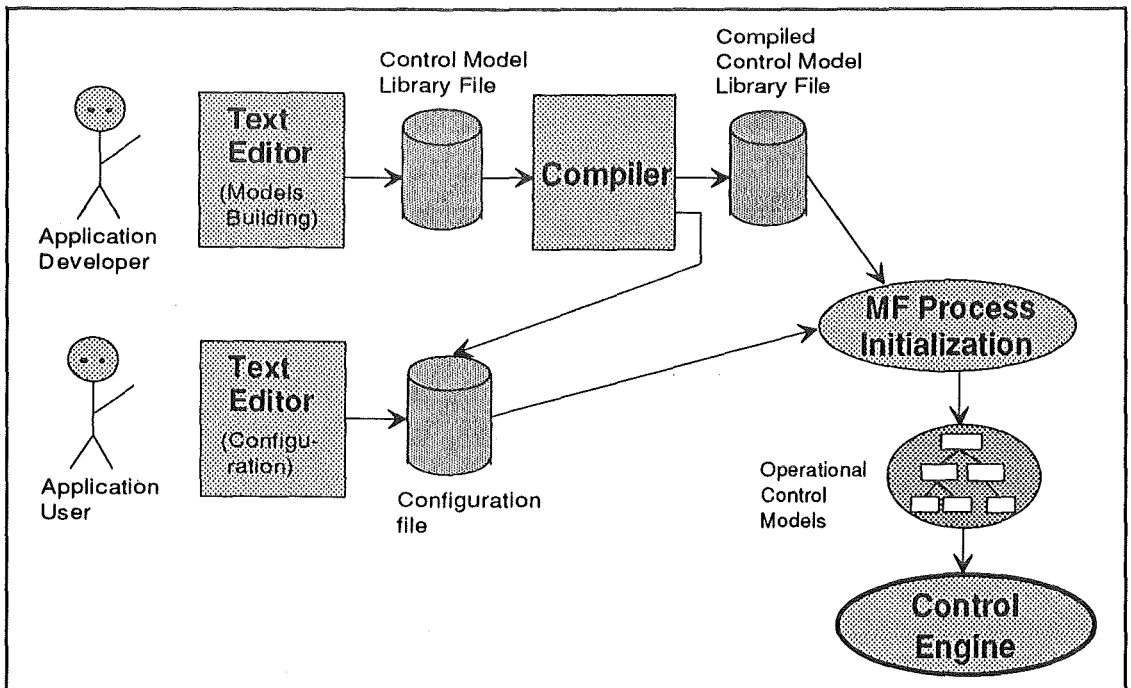


Fig. 4.3 The Processing of the Control Model Library

The application developer builds the Control Models by use of a text editor. The Control Model Library File, consisting of a number of Control Models, is stored as a text file in a storage media. A compiler is used to check the syntax and compile the text file into a binary file with a specific format. In alliance with the compiled Control Model Library File, a configuration file is created additionally. The configuration file contains a table of all the Control Models' names existing in the Control Model Library File. The application user can use a text editor to update this table and select the Control Models needed for the session. When the MF process is initiated, the required Control Models will be extracted from the Control Model Library and loaded into the working memory. The reason for this is to avoid loading a large number of Control Models which are not needed in the running of the session.

At the initialization process all the Control Models of the library to be loaded are organized in a hierarchical structure and stored in the working memory. They can afterwards be accessed by the Control Engine at the requests sent by MF-Clients. The basic reason for this type of procedure is to speed up the processing time of Control Models by the Control Engine.

To investigate to what extent the support for model building should be provided, one needs to examine the industrial user requirements with respect to the MF which have been listed in Section 2.2. The concept of *Service Units*, which provides Control Models with a number of closed and consistent functions, leads to a significant simplification of the MF design. Some important control mechanisms for the model representation are discussed below:

- Repeat a sequence of actions:
  This support can be used to construct the control sequence for the Function Model in which some actions should be repeated a number of times or some parts of manufacturing devices should be periodically monitored;
- Branch to undertake a given action on the appearance of a predefined condition:
  This support is for the case when for example, during the execution of a Function Model a problem occurs. In this case, the Machine Front-End must, either automatically or according to intervention by the operator, take the control of the execution of some specified jobs;
- Send request to ask to start a given Service Unit:
  This support is used to send a Protocol Data Unit to the underlying process which provides a number of Service Units;

- Wait to synchronize the actions of several manufacturing devices:
  This support can be used where several manufacturing devices on the network need to take actions synchronously. It can also be applied when it just simply waits for a specified event, before it can continue the next action;
- Timing operation:
  Timing is a very important feature in the manufacturing process. The operation on the timing should be supported. The timing operation can be used for the schedule, synchronization and polling mechanism, etc.

Moreover, it is also important to support the ***priority-driven execution***, particularly in an environment (such as the MF environment) of a manufacturing process in which multiple co-operations should be undertaken. The requests with higher priority should be processed before those with lower priority.

Figure 4.4 depicts the program structure of the Control Model Library. A Control Model Library File can contain an unlimited number of Control Models. However, the number of Control Models to be loaded is limited by the size of working memory. As stated before, a configuration file is provided for the application user to select the Control Models needed.

A Control Model is identified by a unique name. It contains several service blocks such that each block stands for one type of service requests. The number of service blocks in a Control Model depends on the number of service types for an *Abstract Object* to which the Function Model is applied. All the Control Models are grouped by *Abstract Objects* and stored in a Control Model Library File.

The implementation of the Control Model Library is the task of an application developer. He codes the design specification of CIM-OSA *elementary Function Models* (i.e *Machine Functional Operations*) into a program by use of the supported command language. In order to create an example of implementation of *Machine Functional Operations*, like the one shown in Figure 1.10, the *Abstract Objects* and *services* provided by the Machine Front-End should be specified. They are described in Chapter 6 and the example is given in Chapter 7. The next section describes the support of the Control Models implementation and its use.

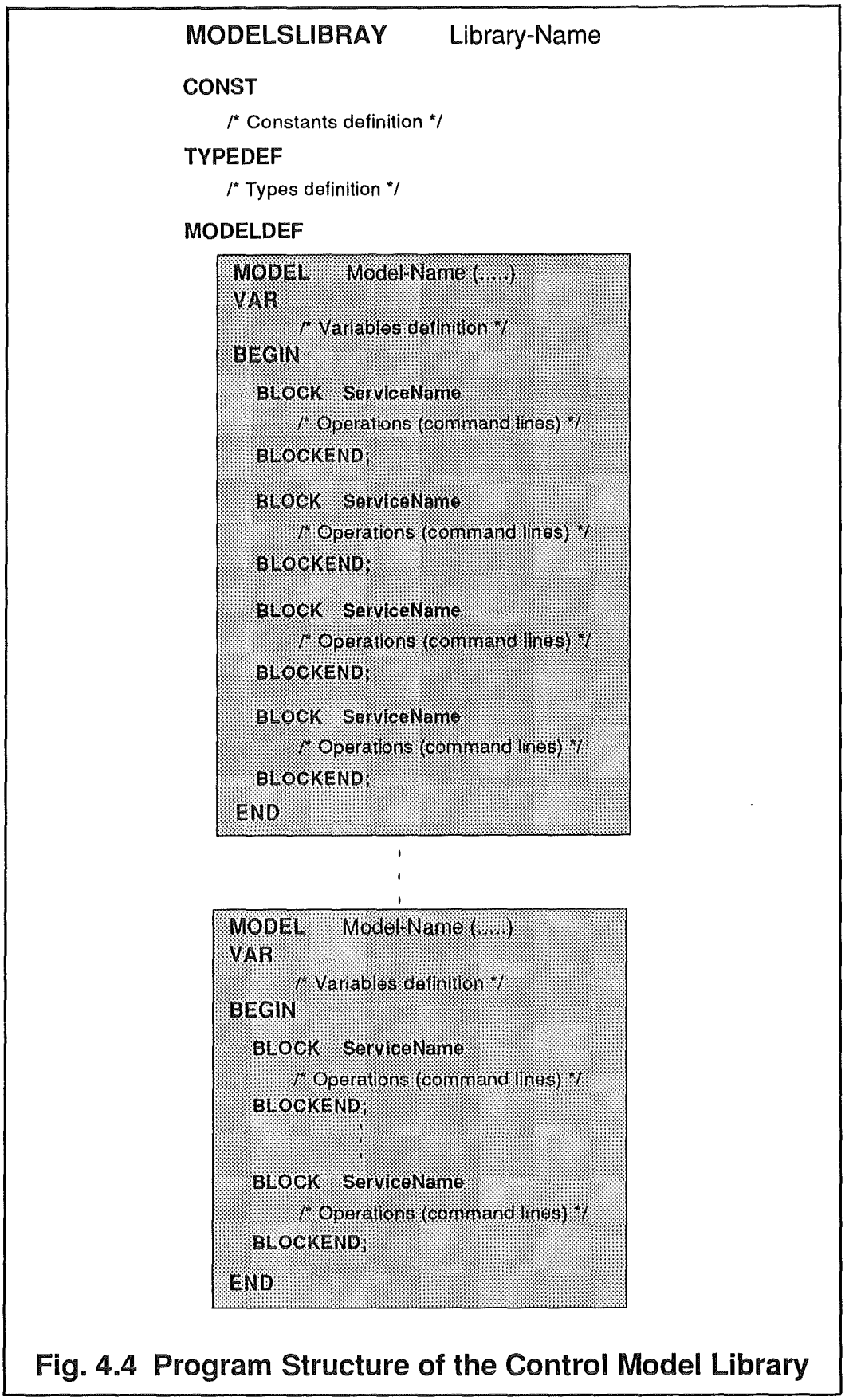


Fig. 4.4 Program Structure of the Control Model Library

4.4) Support of the Control Model Implementation

This section describes some important key elements of the defined command language and shows, by use of examples, how the two industrial user requirements of monitoring and synchronization can be fulfilled. The techniques applied for the handling of the model variables and of the changing message size, the internal representation format of commands and the data types supported for the internal model representation, and the run-time control structure of the Control Model Library are given detailedly in [*Hou93b*].

For the handling of the model variables, a ***Variables Description Block*** (***VDB***) for each Control Model is generated at the MF initialization. The VDB is implemented as an array, each element of the array presents a unique variable. In VDB, a model variable is described by the type, size and value or a pointer to the value block.

The syntax of the command language is defined by an Operation Code and several Operands. The Operation Code specifies the kind of operation. The Operands are expressed in form of variables, constants, logical addresses or *Service Unit Identifiers*. The general format of a command is given below, where the square bracket indicates that the operand is optional.

Opcode [Operand_1] [,Operand_2] [,Operand_3]

A ***Service Unit Identifier*** will be responded to by a ***Service Unit*** implemented in the Service Unit Provider. It is issued by the REQUEST command. The following describes some important key elements of the defined command language which appear in the examples given later in this section.

- ***Request***:This command is used to send a request Protocol Data Unit to the MF-Server. The *SU_Id* is the system-wide unique *Service Unit Identifier* which will be responded to by the underlying *Service Unit Provider*. Both *ReqVar* and *ConfVar* are variables. The *ReqVar* is for the request Protocol Data Unit to be sent to the MF-Server, while the *ConfVar* is for the confirmation Protocol Data Unit to be received from the MF-Server.

  REQUEST SU_Id, ReqVar, ConfVar;

- ***Response***: This command is used to make a reservation for the store of a response Protocol Data Unit to be sent to the MF-Client.

  RESPONSE RspVar;

- ***Wait***: This command is used to wait for a given confirmation of the previously issued request. It can not continue to execute the next command, until the waiting confirmation has arrived.

  WAIT ConfVar;

- ***WaitAll***: This command is used to wait for confirmations of all the previously issued requests. It can't continue to execute the next command, until all the waiting confirmations have arrived. There is no operand for this command.

  WAITALL;

- ***SYSTIME***: This command is used to get the current system daytime value.

  SYSTIME SysTimeVar;

- ***INTERVAL***: This command is used to delay a given time duration.

  INTERVAL DelayType, BaseTime, Interval;

  The *BaseTime* is the absolute daytime value, which can be obtained by the SYSTIME command. The *DelayType* identifies the effects of the command on the time delay and has an enumerate type with three variants which are:

  a) ABSOLUTETIME which is used in the case when the MF should wait until the absolute daytime specified in the *BaseTime* is reached. If the specified daytime has run out, no delay will be performed. The value of *Interval* is ignored;

  b) REFSYSTIME which is used when the Machine Front-End waits for a time delay specified in the *Interval*. In this case the reference time is the current system daytime. The value of *BaseTime* is ignored;

  c) REFBASETIME which is used when the Machine Front-End waits until a specified timepoint is reached. The timepoint is calculated by use of the *Interval* and the *BaseTime*. After the execution of the command, the *BaseTime* will be replaced by the sum of the previous *BaseTime* and the *Interval*.

The supported commands above can achieve the functions of both the monitoring of the shop floor devices and the synchronization of manufacturing devices. These two functions are the important user requirements in the manufacturing process. For example, in an assembly or production line, it is a rule that the operator needs to know the status of shop floor devices. In this case, the Machine Front-End should send service requests to the shop floor devices in a given time interval to retrieve

their status data. A Profile of the example below outlines the monitoring function by reading the status of two machines in 5-second intervals.

```
MODEL  M_Monitor_Example(....)
VAR
    /*variable definition */
BEGIN
    BLOCK  START(...)
        SYSTIME  BaseTimeVar;  /* get the current system daytime in sec.*/
        REPEAT
            /* request to read status data of machine 1 */
            REQUEST   SU_a_Id,      Req1_Var,      Conf1_Var;
            /* request to read status data of machine 2 */
            REQUEST   SU_b_Id,      Req2_Var,      Conf2_Var;
             /* delay 5 seconds */
           INTERVAL   REFBASETIME,  BaseTimeVar,   5000;
        UNTIL  (condition is TRUE);
                                .
    BLOCKEND;
                                .
                                .
END;
```

The synchronization of activities of several shop floor devices needs more sophisticated design than above. The example below describes the handling of synchronization of two robots activities. Suppose that due to the overlap of work area by both robots, robot-A can screw a part onto a workpiece only when robot-B has placed the part on Assembly and moved the arm back. Similarly, robot-B can place a part on Assembly only when robot-A has screwed the previous part onto the workpiece and moved the arm back. A profile of the Control Model is given below which assumes that the programs on both robots have already been loaded.

```
MODEL  M_Synchron_Example(....)
VAR
    /*variable definition */
BEGIN
    BLOCK    START (...)
       /* request robot-B to pick a part from Pallet */
       REQUEST   SU_a_Id,      Req20_Var,      Conf20_Var;
```

```
        WAIT        Conf20_Var;
        /* request robot-B to place the part on Assembly */
        REQUEST   SU_b_Id,     Req21_Var,     Conf21_Var;
        WAIT        Conf21_Var;
        REPEAT
          /* request robot-A to screw a part onto the workpiece */
          REQUEST   SU_c_Id,     Req10_Var,     Conf10_Var;
          /* request robot-B to pick next part from Pallet */
          REQUEST   SU_a_Id,     Req20_Var,     Conf20_Var;
          /* wait till robot-A has finished the screwing and moved the arm back */
          WAIT        Conf10_Var;
          WAIT        Conf20_Var;
          /* request robot-B to place the part on Assembly */
          REQUEST   SU_b_Id,     Req21_Var,     Conf21_Var;
          /* wait till robot-B has finished the placing and moved the arm back */
          WAIT        Conf21_Var;
        UNTIL (condition is TRUE);
    BLOCKEND;

    BLOCK   STOP (...)
              .
              .
END;
```

Another example for the synchronization can be found in the COMETOS system which was developed by the *Institute for Applied Informatics* [*Lawo90*]. The *COMETOS* stands for the *Coordinate Measuring and Tooling System*. It is a highly flexible handling system for tooling of large casting. The COMETOS system is designed as a manufacturing cell. It consists of three robots which are for cutting, measuring and deburring of workpieces. They are connected by an Ethernet. In the COMETOS system, before the deburring robot can be set into the status of deburring, it should be taught by the measuring robot driven by an operator. The teaching of the deburring robot is realized through an interaction (synchronization of activities) of both robots. In this teaching phase, the measuring robot reads the coordinate data of its moving path on a workpiece and sends this data to the deburring robot. With these coordinate data the deburring robot corrects its moving path. A Control Model for this cooperation is given in Chapter 7, which is also used for the MFO-30 'Control' of the Domain Process Model shown in Fig. 1.10.

5) The Control Engine

The ***Control Engine*** is the second fundamental part of the *Machine Front-End (MF)*. It contains the generic control mechanism such as the arrangement for the incoming messages, the management of the *MF Abstract Objects* and the processing of the *Control Model Library*.

In addition to the basic mechanisms which are used in operating systems [*PeSi84*, *XuPa90*], the *Control Engine* of the Machine Front-End has the following features:

- it has a characteristic similar to the Virtual Manufacturing Device [*ISO90*] for the interaction with its clients;
- it is an intelligent interface which co-operates with multiple clients and servers by use of Protocol Data Units for their data exchange;
- it is an application program to the CIM-OSA users (process modeller, implementor of CIM-OSA models, and operator) such that it is transparent to those users;
- the Machine Front-End inside the CIM-OSA framework is fully portable and therefore supports the open systems architecture and can be applied to any CIM-OSA Modules;
- the Integrating Infrastructure which includes the Machine Front-End is actually the highest layer, and may use alternative underlying lower layers such as MMS, FTAM, RPC and other distributed operating system functions. Figure 5.1 gives an overview on those design alternatives.

The *Control Engine* includes three service modules: the Receiving Service Module, Indication Service Module and Transformation Agent. These service modules and the two pending queues are described in this chapter, and the specification can be found in [*Hou93b*].

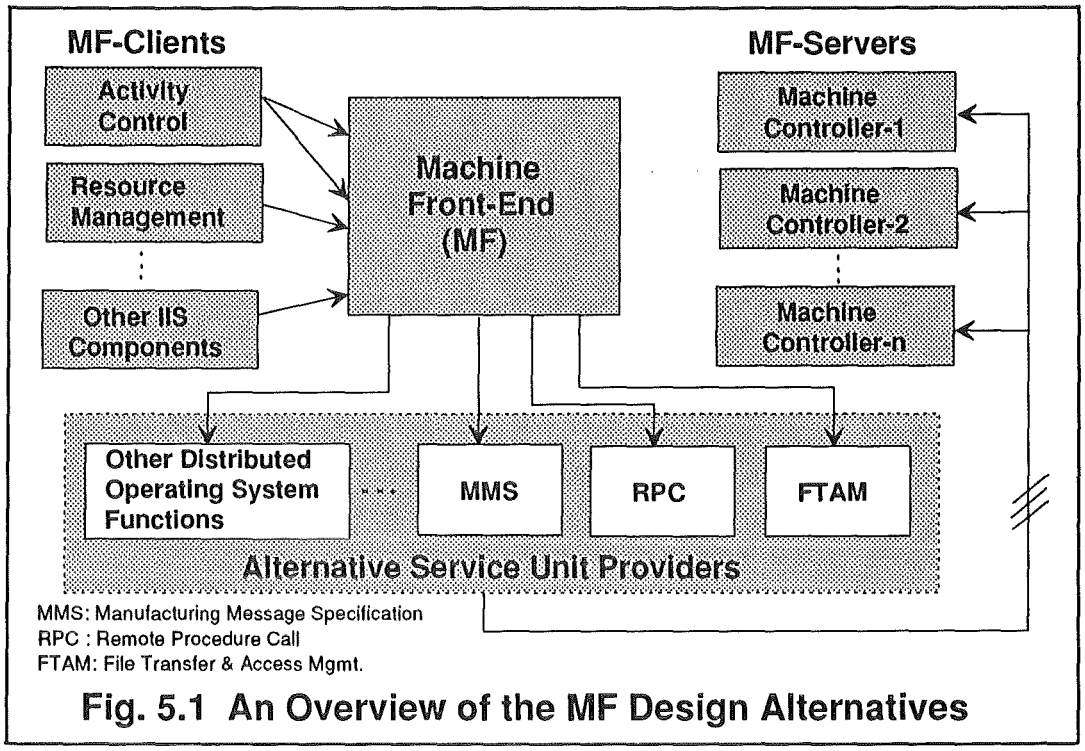


Fig. 5.1 An Overview of the MF Design Alternatives

5.1) Transformation Agent

The ***Transformation Agent*** is the central element of the *MF Control Engine*. For each new pending indication, an instance of ***Model Execution Control Object*** (***ModelExecCtrl***) will be created by the *Indication Service Module*. The Transformation Agent then uses this instance data to process a Control Model.

ModelExecCtrl contains all information for the Transformation Agent to execute a Control Model. It includes the location of the model program codes and the Variables Description Block, MFO identifier, current program counter of the Control Model, flag for the model execution control, a list of waiting confirmations, etc. It is initialized with the data from the run-time control structure of the Control Model Library which represents the information tree described in section 4.1.

In this MF design, each service request issued by the Machine Front-End is labelled by an invoke identifier. It is a unique number given by the Machine Front-End and can be used to recognize the confirmation which the previously issued request is responded to. Therefore, a waiting confirmation can be marked by both the confirmation and the MF invoke identifiers. Within the execution of a Control Model, all the waiting confirmations are registered in an array such that each array element

represents a waiting confirmation. The array element will be removed, when the confirmation arrives.

For each active MF-Service Indication there is an instance of *ModelExecCtrl* for the control of its specified MFO execution. All the *ModelExecCtrl* instances are put into the ***Model Execution Control List*** (***ModelExecCtrlList***) and indexed by their priority. The priority has a data type of integer with two-bytes defined as follows:

- the higher byte contains the MFO priority issued by the MF-Client, and
- the lower byte holds the priority of the Machine Front-End operating on the applied *Abstract Object*. This priority is assigned by the Machine Front-End. For example the priority order of the Machine Front-End operating on MF-OC (see Fig. 6.4) is Load (highest), Unload, Terminate, Stop, Start and then the Initialize Service (lowest).

Figure 5.2 shows the *ModelExecCtrlList* with the associated Control Models and *Variables Description Blocks*. According to the *Abstract Objects* described in Chapter 6, a particular service block of a Control Model may have to be started internally, when the execution for a MF-Service Request is completed. For example, in MF_OC the service block for Initialize-Service should be started after the execution of the service block for Start-Service has been finished or terminated. In this case, an instance of *ModelExecCtrl* describing the service block for Initialize-Service will be inserted into the *ModelExecCtrlList*.

In addition to the support for the priority-driven processing, a *ModelExecCtrl* contains a flag, *ExecCtrlFlag*, for the control of the model execution. The Transformation Agent can use the *ExecCtrlFlag* to determine which command of the Control Model it should process to, before it jumps to the next Control Model. The *ExecCtrlFlag* has an enumerate type with four variants: *EXEC_TOEND*, *EXEC_ONESTEP*, *EXEC_UNTILNEXTWAIT*, *EXEC_NOTHING*.

EXEC_TOEND indicates that the Transformation Agent should complete the whole Control Model before it continues to process other Control Models. It is particular useful for the time-critical operations. *EXEC_ONESTEP* is for the execution of only one command. *EXEC_UNTILNEXTWAIT* tells the Transformation Agent to process the Control Model until it meets the WAIT or WAITALL command. *EXEC_NOTHING* means that the process of the Control Model is found in a stop state. In this case the Transformation Agent does not need to process the Control Model.

Other attributes shown in Figure 5.2 are: *Client_Id* is the identifier of a MF-Client; *MfoCode* is the identifier of a MFO; *CodeList* is a pointer that points to a location which in turn contains several pointers, each of them points to a service block of the Control Model; *VDB* is a pointer to the location of the Variables Description Block associated with the Control Model; and the *PC* is the model program counter which indicates the current process step on the model program code.

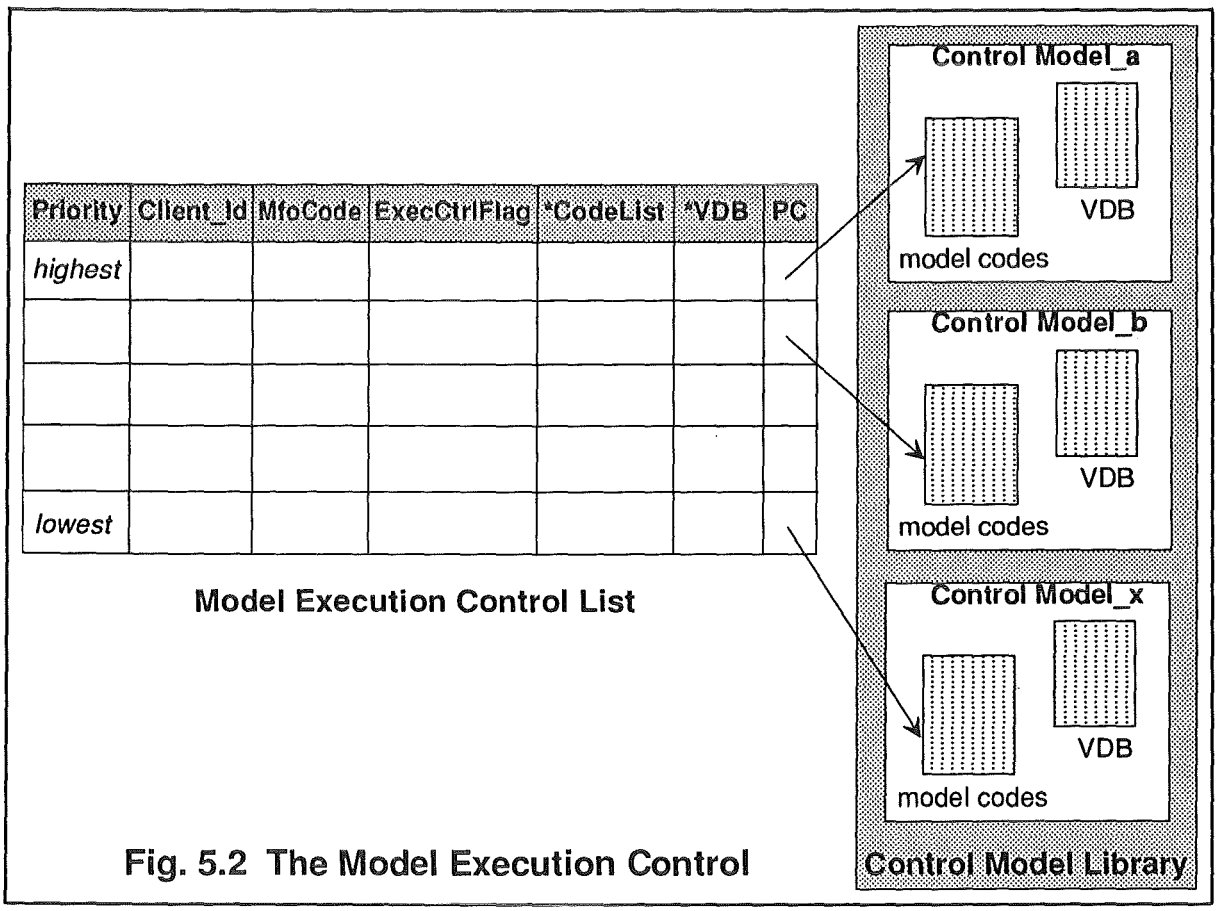


Fig. 5.2 The Model Execution Control

By use of the *Model Execution Control List*, the Transformation Agent can process a number of Control Models concurrently. It is able to deal with several CIM-OSA models in a quasi parallel way. The Transformation Agent interprets each instruction of a Control Model and takes the corresponding actions. As soon as the Transformation Agent has completed a service request, it will change the state of the associated *Abstract Object Instance* (cf. Section 6.3).

The Transformation Agent consists mainly of two program modules which are called ***Model Execution Control Unit*** (***ModelExecCtrlUnit***) and ***Command Execution***

Unit (***ComExecUnit***). The *ModelExecCtrlUnit()* examines the *Model Execution Control List* and processes a Control Model by use of the references to the model codes and the Variables Description Block. It uses the *ExecCtrlFlag* to decide how to proceed in the Control Model. It begins to process the Control Model to which the *PC* points. It calls the *ComExecUnit()* to execute the commands of the Control Model and takes the responsibility for the actualization of the *ModelExecCtrlList* and of the state of the associated *Abstract Object Instance*. Each time when the Transformation Agent is called, it will process all the Control Models through the *ModelExecCtrlList* at one time.

The ***Command Execution Unit*** (***ComExecUnit***) is called by *ModelExecCtrlUnit()*. It interprets each command of the Control Models which are stored in the specified format. In response to each command, a procedure will be called to perform the required operations. All the procedures used to respond to commands have two input parameters: a pointer to the instance of *ModelExecCtrl* and an another pointer to the address of the command to be processed. They have the same return value as those from *ComExecUnit()*.

5.2) The Receiving and Indication Service Modules

CIM-OSA system is a distributed CIM system. The main features of a distributed system are: 1) multiple processes installed in the hardware elements are allowed to do their activities concurrently; 2) the message exchanges between these processes can be executed independently.

The Integrating Infrastructure (IIS) achieves the functions of a distributed (concurrent) system by taking the controls of all processes connected to the system. In addition to the external processes, the Integrating Infrastructure itself requires several processes for its implementation. In the CIM-OSA concepts, all the data communications between the productive IIS service processes (i.e. Business, Front-End, and Information Services) are done through the process of Communication Services. Virtually, the Communication Services acts as a server process within the client-server relationships of the IIS processes. During data exchange the Communication Services will not react to a pending request from its client. This means that, e.g. if a service request is sent by the Business Services to the Machine

Front-End, the Communication Services will not inform the Machine Front-End to pick up this request, it just puts this request message into a queue.

To satisfy this CIM-OSA concept, the Receiving Service Module is designed to be periodically called by the Machine Front-End. It is not designed to be triggered by the Communication Services when the Communication Services receives a message for the Machine Front-End.

The messages queued in the Communication Services for the Machine Front-End can be either the MF-Service Requests from the MF-Clients or the service responses from MF-Servers. If the message is a MF-Service Request, an indication data unit will be constructed and then inserted into the *Pending Indication Queue*. If it is a service response, it will be linked with the previously issued service request and put into the *Pending Request Queue*. The key for searching the service request can be the *Invoke Identifier* issued by the MF when that service request was sent.

The *Indication Service Module* is responsible for processing *MF-Service Indications* which are placed by the *Receiving Service Module* in the *Pending Indication Queue*. It reads indication messages from the Queue and validates these messages. If any unknown or invalid information in an indication message is detected, an error response will be immediately sent back to the MF-Client. Otherwise, it will generate and initialize an instance of the *Model Execution Control Object* and put it into the *Model Execution Control List*.

Indication data units are distinguished by both the client identifier and the indication identifier. The client identifier is a CIM-OSA system wide unique identifier which is given by the Communication Services, and the indication identifier is the same as the invoke identifier issued by the MF-Client when the request was sent. There are four states defined for the description of an indication status. These are: 1) IND_NEW for an indication which is not yet processed by the MF; 2) IND_RENEW for an indication which was stopped and is now restarted; 3) IND_ACTIVE for an indication being in processing; 4) IND_COMPLETED for a completely processed indication.

In fact, the Machine Front-End communicates only with two processes. These are the Communication Services for the transfer of *MF-Access Protocol Data Unit*, and the underlying Service Units Provider for the *MF-External rotocol Data Unit*. In the structure of the Communication Services given by AMICE [*Hou 92*], the communication with the underlying Service Units Provider can be arranged through

the '*Transfer Agent*', a part of the Communication Services. In the MF design specification, there are four interface functions used by the Machine Front-End for sending and receiving messages, which are:

- S_RecvMsg() to receive messages from the Communication Services;
- S_SendMFMsg() to send messages to the Communication Services;
- S_RecvSURsp() to receive messages from the underlying Service Units Provider;
- S_SendSUReq() to send messages to the underlying Service Units Provider.

To conform to the CIM-OSA concept, these four interface functions should use the callable functions of the System Wide Exchange (SE), e.g. SE_TELL() for sending and SE_ACCEPT() for receiveing messages [*CIMO90-C5-3000*]. These interface functions are system-dependent. When the implemented Machine Front-End is going to be transferred into a different platform, these interface functions should be adapted to the new environment.

5.3) MF Pending Queues

As given in Figure 3.5, two queues are used to track the outstanding messages: the ***Pending Indication Queue*** (***MFINDPEND***) for the pipeline of the pending *MF-Service Indications* and the ***Pending Request Queue*** (***MFREQPEND***) for the registration of the issued service requests.

Figure 5.3 shows the link between these two queues. In order to register all the issued service requests within a Control Model for a specified indication, these two queues are linked by a pointer. Each confirmation data received from MF-Server is joined with its previously issued service request.

Each indication data unit of the *Pending Indication Queue* contains a *MF Access Protocol Data Unit*. Similarly, each request data unit of the *Pending Request Queue* holds a *MF External Protocol Data Unit* which includes the Service Unit-specific request data and confirmation data. The request and confirmation data are registered for use by the subsequent service requests of a Control Model. After a response is sent back to the MF-Client, the indication data unit and all the request and confirmation data units for the indication are removed.

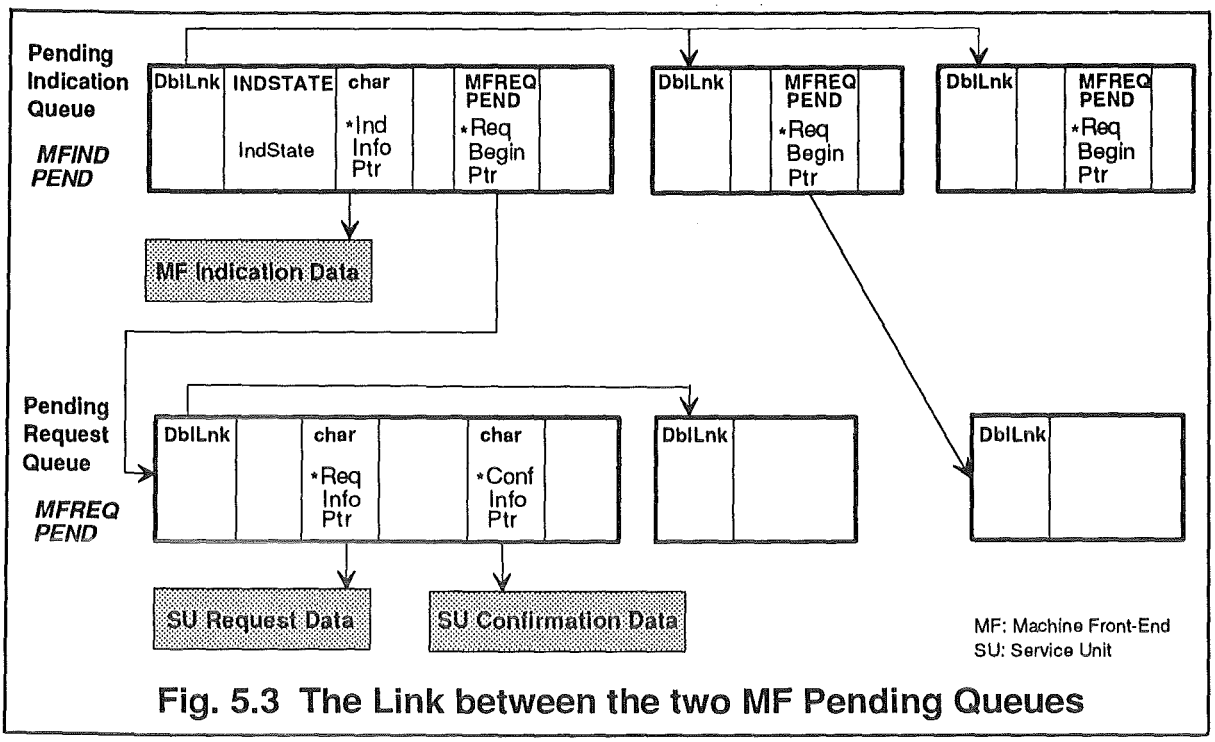


Fig. 5.3 The Link between the two MF Pending Queues

In order to ease the removing or adding of a data unit from/to a queue, a double-link is used, which is represented by the attribute *DblLnk*. The *IndState* indicates the state of the indication data unit which can be one of the IND_NEW, IND_RENEW, IND_ACTIVE or IND_COMPLETED described in the previous section. The *IndInfoPtr* is a pointer which points to the indication data unit, and the *ReqBeginPtr* is used to link the issued service requests to the MF-Server. The service request data (i.e. SU Request Data) is pointed to by the *ReqInfoPtr*, while the confirmation data (i.e. SU Confirmation Data) is pointed to by the *ConfInfoPtr*. These two attributes belong to the Pending Request Queue of MFREQPEND.

The ***parallel processing*** of multiple Function Models is considered to be one of the important keys for the success of the MF design. This can be achieved by the *Control Engine* specified so far. The Machine Front-End doesn't need to complete a *MF-Service Indication* at once. It can serve another pending *MF-Service Indication*, even if there are already several *MF-Service Indications* being processed. This means that the MF is able to process a number of CIM-OSA models concurrently.

Similarly, a number of service requests can be issued immediately one after another. If it is not demanded, the Machine Front-End doesn't need to wait for the response. The Machine Front-End can recognize the incoming confirmation which the previously issued service request has responded to. In this sense, the Machine

Front-End is able to deal with a number of service requests/confirmations within a CIM-OSA model in a parallel way.

Moreover, for the time critical operations in a distributed system, the *Control Engine* supports the ***priority-driven operation*** and the ***control-oriented execution.*** This means, that the Function Models with higher priority will be processed prior to those with lower priority, and the execution control of a Function Model can be set to a state such that it should be completed before other Function Models can be processed.

6) MF Abstract Objects and Protocols

This chapter mainly addresses the interaction between the *Machine Front-End (MF)* and the other components of the Integrating Infrastructure. It discusses the ***Object Modelling Technique*** used by the international standard *Manufacturing Message Specification (MMS)*. The proposed approach of the MF design applies this technique to specify the MF capability. Based on the guidelines given by CIM-OSA, an Abstract Object for the operational control is defined and from this object the supporting services are specified. Furthermore, the management of the Abstract Objects, the definition of Access, External Protocol and some Service Units are described.

6.1) Object Modelling Technique

The data communication between two co-operating partners in the CIM-OSA world is based on the ***Client-Server Architecture***, in that the communication must be able to service any request and response between the two partners. A partner can act either as a client by sending a service request, or as a server on receipt of the service request.

So far, the approach to the MF design was introduced. Both the support for the building of the *Control Model Library* and the *Control Engine* were specified. The separation of the *Control Model Library* and the Control Engine has a crucial importance for this approach. It not only solves the implementation problem of the application-specific control knowledge, but also provides a great flexibility for the modelling of enterprise activities.

Up to now, the interactions of the Machine Front-End with its clients (e.g. Business Services, Human Front-End, etc.) and the support for these interactions were not addressed. Before we are going to deal with these problems, it is worth examining the ***Object Modelling Technique*** which was used for the widely accepted *Manufacturing Message Specification (MMS)* [*ISO90*]. The MMS is intended for machine to machine communication, i.e. the communications between computers and programmable devices like robots, NC-machines, Programmable Logic Controllers and realtime process controllers [*Holl91*].

The modelling technique used in the MMS describes the Abstract Objects, the characteristics of such objects, and the operations on those objects. Any manufacturing device can be modelled by an entity, called a ***Virtual Manufacturing Device (VMD)*** or a ***MMS-Server***, which contains several Abstract Objects and provides a number of services. A ***MMS-Client*** can use these services to manipulate these Abstract Objects, that means to change the behaviour of the VMD.

The MMS provides 84 basic services for operating on MMS objects. The MMS objects are grouped into 12 classes: Named Variable, Scatter Access, Named Variable List, Named Type, Semaphore, Event Condition, Event Action, Event Enrollment, Journal, Domain, Program Invocation and Operator Station Object. These objects belong to three scopes, called Application Association (AA), VMD and Domain Scope. Figure 6.1 gives an outline of MMS services which are grouped into 10 sets.

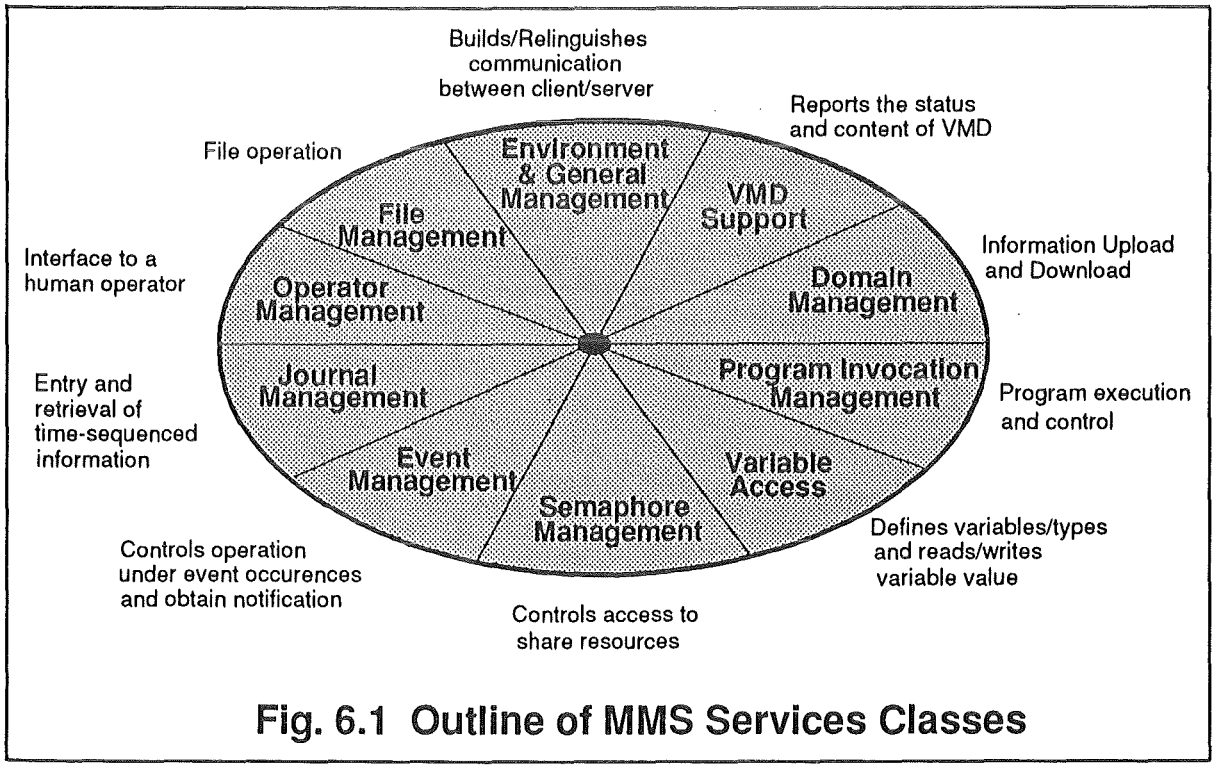


Fig. 6.1 Outline of MMS Services Classes

In MMS, services which are used to manipulate a MMS object are defined on the basis of the ***Object State Transition Diagram***. Each service is described in a tabular form with their arguments and results.

Besides the Variables and the Types, all the other objects are modelled by the *Object State Transition Diagram*. It describes the major states and intermediate states. The intermediate states exist only between an indication primitive and the response primitive or between a request primitive and a confirm primitive. While these states are transient, the MMS object may be in this state for some period of time. The major state exists after processing a response primitive or a confirm primitive. A major state can also be an intermediate state.

Figure 6.2 shows the basic elements of a *State Transition Diagram*. A box indicates an object state. A line indicates the transition. S1, S2 and S3 are major states, D1 is an intermediate state. After the process receives an indication, it will change the state of the object from S1 to D1.

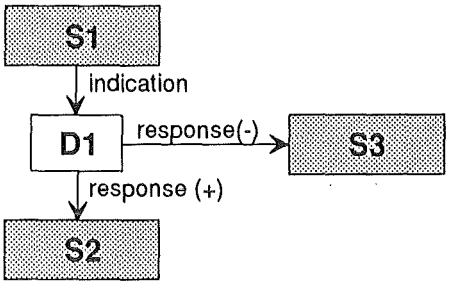


Fig. 6.2 Basic Elements of a State Transition Diagram

It is also worth analyzing the MMS software product to get a more detailed understanding of the MMS and the implementation technique. The analysis of the object structure of *MMS-EASE (MMS-Embedded Application Service Element)*, the most notable product of the MMS implementation [*SISC90*], is given in Figure 6.3. MMS-EASE provides several local logical channels for each network node. Each channel can be used by a user application. The application processes on each network node must be assigned to the local channels. Then, the associations between application processes can be established via the connections of the logical channels. In a multi-tasking environment, it is possible at one time to establish a number of applications' associations for a physical connection and to execute the data exchange within each association simultaneously and independently.

In MMS-EASE, a logical channel represents an association with another process and is modelled by a *Channel Information Object*. It contains all information about the

established association of two application processes, e.g. the current state of the channel, the selected presentation context (MMS core or Companion Standard), the local and the remote application process names, the type of the request service and response service, a link to the *AA-specific Domain Object* and a link to the selected *VMD Control Object*, etc.

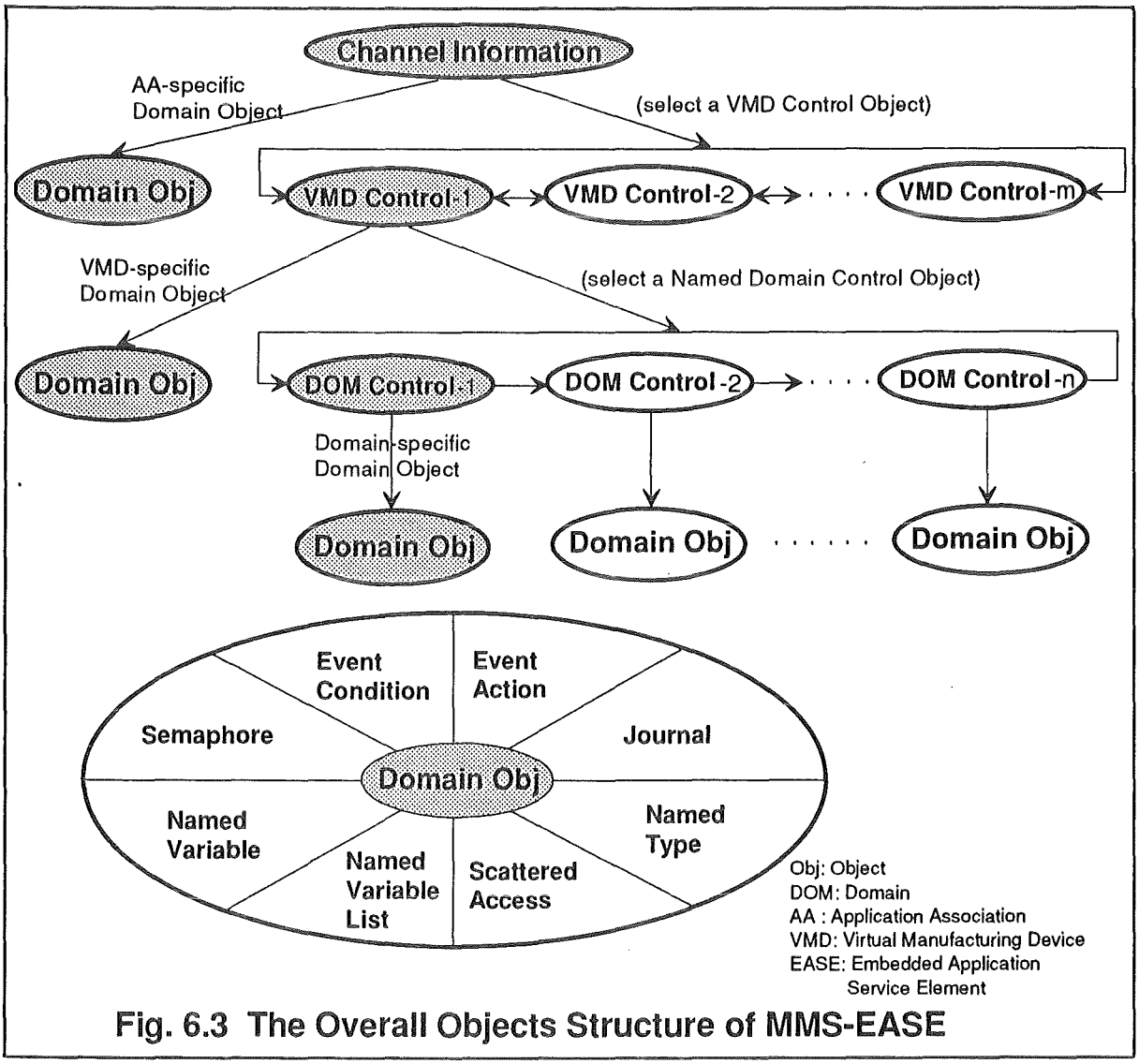


Fig. 6.3 The Overall Objects Structure of MMS-EASE

In MMS-EASE, a *VMD Control Object* (VMD Control) is used to represent a VMD. It is linked with its preceeding and succeeding *VMD Control Objects*. This means that several VMD's can be established for a server application. This object contains the VMD-specific information: a VMD-specific Domain Object, pointers to the Domain, Program Invocation, Operator Station and Named Domain Control object, etc.

A VMD may contain several *Domain Objects* such that each of them is managed by a *Named Domain Control Object* (DOM Control). A Named Domain Control Object contains the domain-specific information, such as the domain name, the domain state and an Domain Object, etc.

A *Domain Object* of MMS-EASE includes a number of MMS-objects, such as Named Variable, Scattered Access, Named Variable List, Named Type, Journal, Semaphore and Event. In MMS, most of these objects are ordered according to all the three scopes: AA (Application Association), VMD and Domain Scope. In conformance with MMS, MMS-EASE allows the creation of the Domain Objects for any of the three scopes.

In MMS-EASE, all the MMS objects are implemented as dynamic objects. It means that they come into existence during the course of operation of the VMD. They can be created either on the request service primitive sent by the client or through the local service support within the server application.

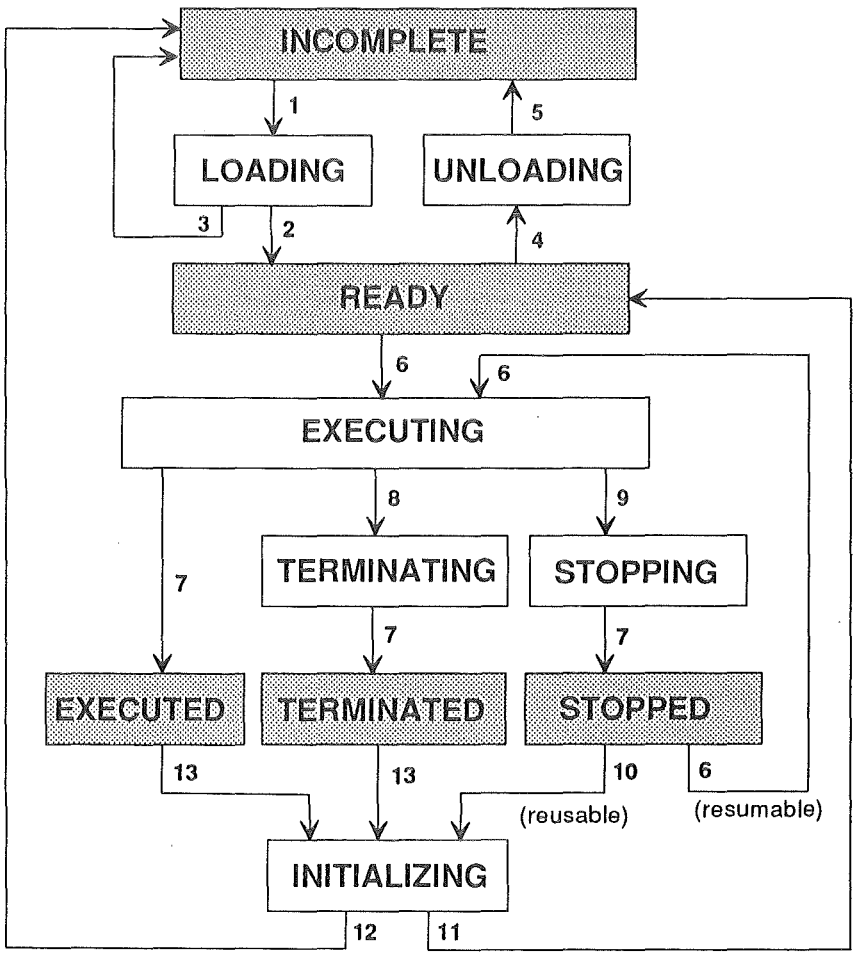
In the client-server architecture of the CIM-OSA IIS environment, the Machine Front-End (MF) can be viewed as an entity which is similar to a MMS-Virtual Manufacturing Device. The Machine Front-End can be described by several Abstract Objects, each object represents a subset of the MF capability and is used for a specific purpose. It is important to point out that the Abstract Objects defined for the Machine Front-End are mainly used to specify the supporting services and the procedures of these services called by its clients. It is not the purpose to map these Abstract Objects to a real device in the way of MMS.

6.2) Specification of MF Abstract Objects and Services

The Machine Front-End should provide its clients with several sets of services for the interactions. Each set of services is defined to operate on an *Abstract Object* representing a subset of MF capabilities. The *Abstract Objects* and the supporting services required for the Machine Front-End are dependent on the requirements of all the MF-Clients (other components of CIM-OSA Integrating Infrastructure).

CIM-OSA has given a guideline for the definition of a set of services for the control of the MFO execution [*CIMO90/C5-3000*], which includes Load, Unload, Start, Stop,

Terminate and Initialize. In conformance with this definition and by means of the *Object Modelling Technique* described before, an *Abstract Object* is specified. This *Abstract Object* is called the ***MF-Operation Control Abstract Object*** (***MF_OC***). It is modelled by the *Object State Transition Diagram* shown in Figure 6.4 and contains the information of states and transitions. The MF_OC exists at and is manipulated by the Machine Front-End on behalf of a MF-Client, and describes the behaviour and interactions between the Machine Front-End and its clients.



Transition lines for the model are:

1 - Load.indication
2 - Load.response (+)
3 - Load.response (-)
4 - Unload.indication
5 - Unload.response (+/-)
6 - Start.indication
7 - Start.response (+/-)
8 - Terminate.indication
9 - Stop.indication
10- Initialize.indication
11- Initialize.response (+)
12- Initialize.response (-)
13- MF Internal transition

Fig. 6.4 MF_OC State Transition Diagram

In the MF_OC *State Transition Diagram*, the intermediate states are indicated by the blank-boxes, whereas the major states are shown in the shadow-boxes. When an object stays in an intermediate state, the Machine Front-End is co-operating with its servers by use of Control Models. For example, after the Machine Front-End has received a Start indication, it changes the MF_OC state from the major state of READY into the intermediate state of EXECUTING. During the phase of EXECUTING state, the Machine Front-End is interacting with its servers by use of the Control Model. As soon as the Machine Front-End has accomplished this task or met an error, it sends a Start response message back to its client and then changes the MF_OC state into the major state of EXECUTED. At the EXECUTING state, the MF-Client can interrupt or stop the MFO execution. If the execution of a MFO is stopped, it can be continued from the STOPPED state or re-initialized.

For a specified MFO execution requested by a MF-Client, there should exist an instance of the appropriate *Abstract Object* for controlling and monitoring the MFO executions. An instance of MF_OC comes into existence only if the machine control program which is required for the MFO execution is successfully installed in the MF-Server. The instance will be removed after the program has been unloaded or explicitly declared to be unusable.

Both the Load and Unload services are used to preprare the resources (i.e. machine control programs) for a MFO execution. The loading procedure checks if the Program Units for the MFO execution is properly installed in the MF-Server. If it is not, it will begin an installation process. In contrast, the unloading procedure is to remove the Program Units from the server. According to the CIM-OSA concept these two services are issued by the Resource Management of the Business Services. The content of Resource Management is still not defined, so we are not concerned with these two services in this work. Therefore, it is supposed that all the machine control programs are ready for the execution before the Activity Control of the Business Services issues a Start service. The description of MF_OC is given as follows:

Object: MF_OC
  Key Attribute: MF_OC Instance Identity (Key=Client_Id + Server_Id + MfoCode)
  Attribute: State (INCOMPLETE, LOADING, UNLOADING, READY, EXECUTING,
    EXECUTED, STOPPING, STOPPED, TERMINATING,
    TERMINATED, INITIALIZING)
  Attribute: Client identifier
  Attribute: Server identifier
  Attribute: MFO Code

The four services of Start, Stop, Terminate and Initialize are described below.

MF-Start Service:

This service can be called by a MF-Client to start the execution of a specified MFO. The service structure is shown in the table below:

Parameter	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
MFO name	M	M(=)		
Start Argument	U	U(=)		
Result			M	M(=)
Response Event			M	M(=)
Executed State			S	S(=)
Terminated State			S	S(=)
Stopped State			S	S(=)
Result Data			U	U(=)

The parameter types in the service table indicate [*ISO 90a*]:

M - parameter is mandatory for the service primitive;

U - parameter is a user option;

C - parameter is conditional upon other parameters;

- (blank) parameter is never present;

S - parameter is a selection from a collection of two or more possible parameters.

= - this code following one of the codes M, U, C or S indicates that the parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table

Parameter description:

a) Argument:

This parameter conveys the parameters of the Start Service request.

a.1) MFO name:

This parameter, of type Identifier (character string) specifies the MFO that will be started.

a.2) Start Argument:
This parameter is an optional field which may be used to pass data to the Control Model for the specified MFO. It is either a character string or an externally coded value.

b) Result:
This parameter indicates that the service request succeeded or failed, the state of the MF_OC instance and the MFO state.

b.1) Response Event:
This parameter identifies the state of MF_OC instance (EXECUTED, TERMINATED or STOPPED).

b.1.1) Executed State:
This parameter identifies the MFO state if the Response Event is EXECUTED. It can be one of the following states: EXECUTED REUSABLE, EXECUTED UNUSABLE, DEADLOCK, LIVELOCK.

b.1.2) Terminated State:
This parameter identifies the MFO state if the Response Event is TERMINATED. It can be one of the following states: TERMINATED REUSABLE, TERMINATED UNUSABLE, DEADLOCK, LIVELOCK.

b.1.3) Stopped State:
This parameter identifies the MFO state if the Response Event is STOPPED. It can be one of the following states: STOPPED REUSABLE, STOPPED UNUSABLE, STOPPED RESUMABLE, DEADLOCK, LIVELOCK.

b.2) Result Data:
This parameter is an optional field which may be used to pass data to the MF-Client. It is either a character string or an externally coded value.

MF-Stop Service:

This service can be called by a MF-Client to arbitrarily interrupt the execution of a started MFO with option to re-start it later. The Stop Service does not have an own response. The service structure is shown in the table below:

Parameter	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
MFO name	M	M(=)		
Stop Argument	U	U(=)		

Parameter description:

a) Argument:
This parameter conveys the parameters of the Stop Service request.

a.1) MFO name:
This parameter, of type Identifier (character string), identifies the started MFO that will be stopped.

a.2) Stop Argument:
This parameter is an optional field which may be used to pass data to the started MFO that will be stopped. It is either a character string or an externally coded value.

MF-Terminate Service:

This service can be called by a MF-Client to terminate the execution of a started MFO at a defined state. In contrast to the Stop service, after the termination, it can not be restarted. The Terminate Service does not have an own response as the Stop Service. Similarly, a *Terminate MFO Indication* is responded to by a Start Response as well. The service structure is shown in the table below:

Parameter	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
MFO name	M	M(=)		
Terminate Argument	U	U(=)		

Parameter description:

a) Argument:
This parameter conveys the parameters of the Terminate Service request.

a.1) MFO name:
This parameter, of type Identifier (character string), specifies the started MFO that will be terminated.

a.2) Terminate Argument:
This parameter is an optional field which may be used to pass data to the started MFO that will be terminated. It is either a character string or an externally coded value.

MF-Initialize Service:

This service can be called by a MF-Client to reset an interrupted MFO (which is found in a Stopped state). The service structure is shown in the table below:

Parameter	Req	Ind	Rsp	Cnf
Argument	M	M(=)		
MFO name	M	M(=)		
Initialize Argument	U	U(=)		
Result (+)			S	S(=)
Result (-)			S	S(=)
Error Type			M	M(=)

Parameter description:

a) Argument:
This parameter conveys the parameters of the Initialize Service request.

a.1) MFO name:
This parameter, of type Identifier (character string), specifies the stopped MFO that will be initialized.

a.2) Initialize Argument:
This parameter is an optional field which may be used to pass data to the stopped MFO that will be initialized. It is either a character string or an externally coded value.

b) Result:
This parameter indicates that the service request succeeded or failed.

b.1) Error Type:
This parameter indicates the error type if the service request failed.

6.3) The MF-Abstract Object Control Structure

This section discusses the concept for the store and management of the *State Transition Diagrams* which describe the Abstract Objects. It defines a control structure for the Control Engine to process the Instances of the Abstract Object.

Before a MF-Client can send a service request (i.e. MFO call) to the Machine Front-End, it has to establish an association with the Machine Front-End. A Machine Front-End can be implemented as a process and identified by a CIM-OSA system wide unique number. By use of the proposed approach a Machine Front-End can deal with multiple MFO calls from several MF-Clients concurrently. Figure 6.5 shows a MF process and its environment, where the Communication Services and the OSI network are not presented.

As stated before, an *Abstract Object* can occur as a number of *instances* such that each instance stands for the control of a specified MFO execution. An *Abstract Object*, in fact, provides the following two functionalities:

- to check if the received indication is valid. If it is detected as being invalid, the Machine Front-End will reject it by immediately sending an error response to its client.

- to indicate the Control Engine to implicitly start a particular service, e.g. to internally start an Initialize-Service automatically, after it has successfully completed the execution of a Start-Service request.

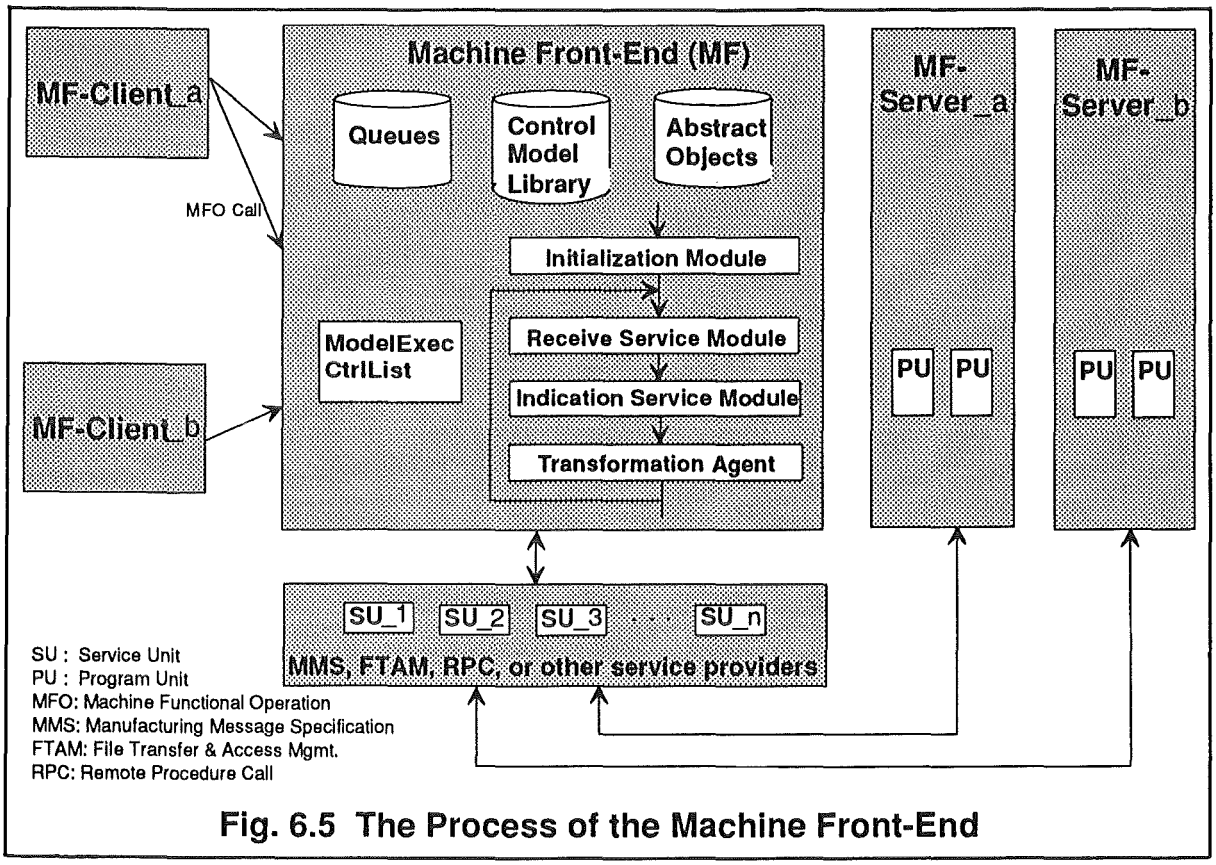


Fig. 6.5 The Process of the Machine Front-End

The state of an ***Abstract Object Instance*** is maintained by the Control Engine. The Control Engine changes the state of an *Abstract Object Instance* when

- it begins to execute a MFO Control Model;
- it has finished the execution of a MFO Control Model;
- it recognizes that a particular service must be "internally" started. This means the forthcomming operation on the specified MFO can not be activated by a MF-Client. The Control Engine must take the responsibility of initiating and running the corresponding service block of the Control Model.

In order to ease the access and maintenance of the instances of *Abstract Objects*, all the *Abstract Objects* and their instances must be well organized. Before the control structure for the *Abstract Objects* is defined, the concept for the management of *Object State Transition Diagrams* will be outlined first.

In the approach of the MF Design, every ***Event*** of an *Object State Transition Diagram* is identified by a predefined constant, called ***Event Identifier***. An *Event* is an action which causes the change of the state, and is shown by the arrow in the diagram. There are three types of *Events* which are indication, response and internal transition (Fig. 6.4). The *Event Identifier* is predefined and unique within the MF. It consists of two parts: one part is an *Abstract Object Identifier*, and the other part is either an indication code, a response code or an internal transition code, which is given below.

Abstract Object Identifier	a) partition-1: Indication code b) partition-2: Response code c) partition-3: Internal transition code

The components of an *Event Identifier*

In fact, the *Event Identifiers for the indication codes* can be used for the definition of the IIS-Service Identifiers as well. This has the advantage of reducing the mapping task between the IIS-Service Identifiers and the Event Identifiers for the indications.

The *Event Identifiers for the internal transition codes* must contain the information such that the Control Engine can recognize that a particular service must be initiated by itself. The identifier of the particular service is itself an *Event Identifier for an indication code*. So that the *Event Identifier for the Internal transition code* consists of the Event Identifier of the particular service and the base address of the partition-3. From this known base address the Control Engine is able to induce the service which should be started by itself.

A ***Transition*** of the *Object State Transition Diagram* describes an Event with a source and a destination state. The number of states and events defined for an *Abstract Object* can be different, therefore for generic purposes, the transitions should be defined in a universal way. This is expressed by the data structure of TRANSITION given below. As an example, the *State Transition Diagram* defined for the MF_OC is represented by the array of *MF_OC_TransTable[]*. Each element of the array has the type of TRANSITION and describes a transition of the MF_OC object.

```
typedef  enum  {SDUMMY}  MF_STATE;     /* dummy element for generic use */
typedef  enum  {EDUMMY}  MF_EVENT;
```

```
typedef struct
    {
    MF_STATE    SourceState;
    MF_EVENT    Event;
    MF_STATE    DestState
    } TRANSITION;

TRANSITION MF_OC_TransTable[ ] =
      {
      {OC_INCOMPLETE, OC_LOAD_IND, OC_LOADING},
      {OC_LOADING, OC_LOAD_RESPPOS, OC_READY},
      {OC_LOADING, OC_LOAD_RESPNEG, OC_INCOMPLETE},
      {OC_READY, OC_UNLOAD_IND, OC_UNLOADING},
      {OC_UNLOADING, OC_UNLOAD_RESP, OC_INCOMPLETE},
      {OC_READY, OC_START_IND, OC_EXECUTNG},
      {OC_EXECUTING, OC_START_RESP, OC_EXECUTED},
      {OC_EXECUTING, OC_TERMINATE_IND, OC_TERMINATING},
      {OC_EXECUTING, OC_STOP_IND, OC_STOPPING},
      {OC_TERMINATING, OC_START_RESP, OC_TERMINATED},
      {OC_STOPPING, OC_START_RESP, OC_STOPPED},
      {OC_STOPPED, OC_START_IND, OC_READY}
      {OC_EXECUTED, OC_INITIALIZE_T1, OC_INITIALIZING},
      {OC_TERMINATED, OC_INITIALIZE_T1, OC_INITIALIZING},
      {OC_STOPPED, OC_INITIALIZE_IND, OC_INITIALIZING}
      {OC_INITIALIZING, OC_INITIALIZE_RESPNEG, OC_INCOMPLETE}
      {OC_INITIALIZING, OC_INITIALIZE_RESPPOS, OC_READY}
      };
```

In order to provide that the state of every *Abstract Object Instance* can be easily accessed and actualized, a control structure of *Abstract Objects* is given in Figure 6.6. This control structure is started with the *MF_AGENT* which contains all the information about the Machine Front-End. An attribute of *MF_AGENT*, *ObjList*, points to all the *Abstract Objects* defined for the Machine Front-End.

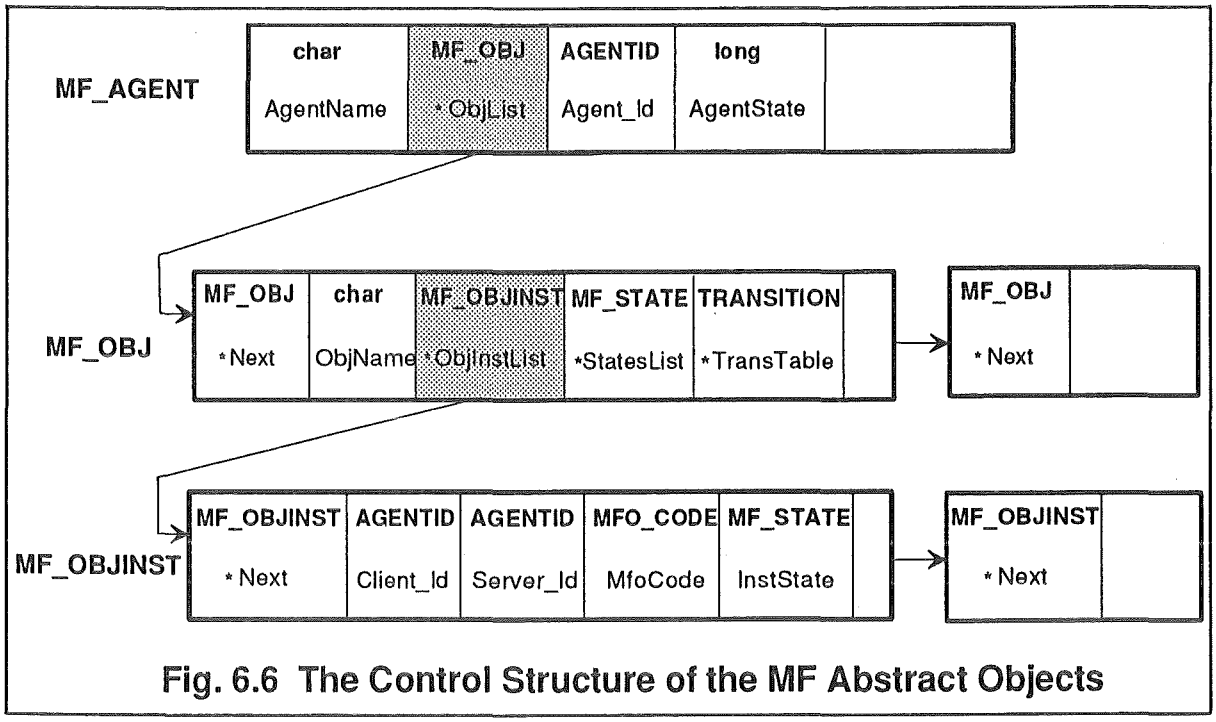


Fig. 6.6 The Control Structure of the MF Abstract Objects

An *Abstract Object* is represented by the data structure of *MF_OBJ* which contains the Abstract Object-specific information, e.g. the Abstract Object name, identifier, all the possible states and the state transition table. An attribute of the *MF_OBJ*, *ObjInstList*, points to all created instances of the *Abstract Object*.

An *Abstract Object Instance* is represented by the data structure of *MF_OBJINST*. It comes into existence only when a Control Model for the specified MFO is going to be started by the Machine Front-End. The Control Engine uses the information stored in the *State Transition Table* (*TransTable* of *MF_OBJ*) to change the state of the *Abstract Object Instance* and, if needed, to initiate a particular service by itself. The instance will be removed when a Control Model has been completed or failed at the execution of the Control Model.

The *Control Structure* of the *MF-Abstract Objects* can be created with some preferred parameters when the system initializes. Thereafter, the MF should be able to add or remove an *Abstract Object Instance* to/from the control structure during run time.

6.4) Interaction between the Machine Front-End and its Clients

The interaction between the Machine Front-End and its client is based on the Client-Server Architecture, which can be characterized by a pair of *complementary interaction models*. In other words, ***the MF-Client should have an interaction model for use which should be consistent with the corresponding Abstract Object and services supported by the Machine Front-End.*** The model is called ***Standard Control Structure*** because it is predefined and stored in the client process for use.

Figure 6.7 describes the ***Standard Control Structure*** used by the **Activity Control** of the Business Services (a MF-Client). It is established in consistence with the ***MF-Operation Control Abstract Object (MF_OC)*** defined before. According to this control structure the Activity Control issues service requests to the Machine Front-End for the control of the MFO execution.

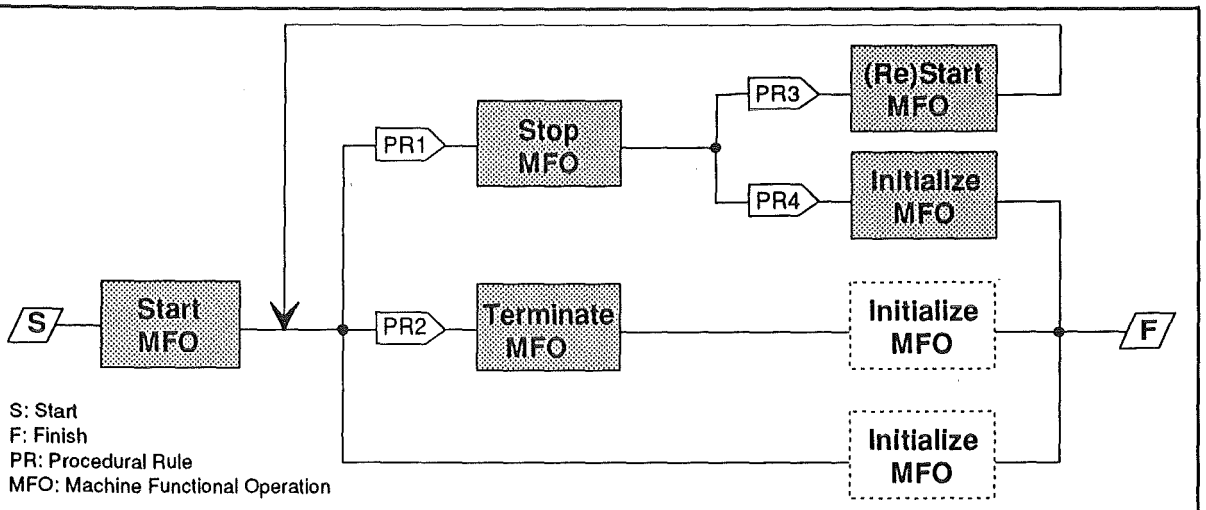


Fig. 6.7 MFO Standard Control Structure

The action in the dotted blank box indicates that the service is not issued by the Activity Control. Internally, it is automatically triggered by the Machine Front-End. For example, after a *Start MFO Service Request* issued by the Activity Control has been sucessfully executed, the Machine Front-End should itself execute the Initialize service block of the Control Model, not because of an *Initialize MFO Service Request* issued by the Activity Control.

Each of the Procedural Rules (PR1 to PR4) is described by a predefined condition. For example, PR1 can be defined as a logical object and initialized with the value FALSE. When an operator stops the execution of a MFO through a Control Panel, PR1 will change to the value TRUE and then the Activity Control will send a Stop Service Request to the Machine Front-End.

6.5) MF Protocols

A protocol is a set of rules (syntactic and semantic) which govern the use of the services by the service users. The message unit for the data transfer between the service user and the service provider is called the *Protocol Data Unit (PDU)*. The PDU contains the protocol information and possible user data, if any.

In order to validate the proposed MF design by means of prototyping, the specification, including some MF Protocols, was defined by a high level language (C-programming language) [*Hou93b*]. The effort on the protocol specification expressed by a neutral notation, such as *ASN.1* (*Abstract Syntax Notation One*) [*ISO 87, GoSp90*], makes sense, only if all the components of the Integrating Infrastructure are well defined, validated and are ready for use.

Figure 6.8 gives some typical sequences of services which can be issued by a MF-Client. These services are provided by the Machine Front-End by means of the MF_OC *Abstract Object* described in Section 6.2.

Two types of MF Protocols are defined for use of the MF-prototype: the *Access Protocol* used by other IIS components to exchange messages with the MF, and the *External Protocol* is used by the MF to co-operate with the machine controllers.

The content of ***MF-Access Protocol*** depends on the services provided by the MF. It should include: 1) identifier of the MF-Client; 2) identifier of the MF; 3) invoke identifier issued by the MF-Client; 4) MF-Service code; and 5) MF-Service specific data unit, e.g. for the Start Service Request or Start Service Response.

The proposed approach applies the concept of *Service Units*. So the content of the ***MF-External Protocol*** depends on the Service Unit used. It should contain: 1) identifier of the MF; 2) indication identifier; 3) invoke identifier issued by the MF; 4) Service Unit identifier; and 5) Service Unit-specific data unit.

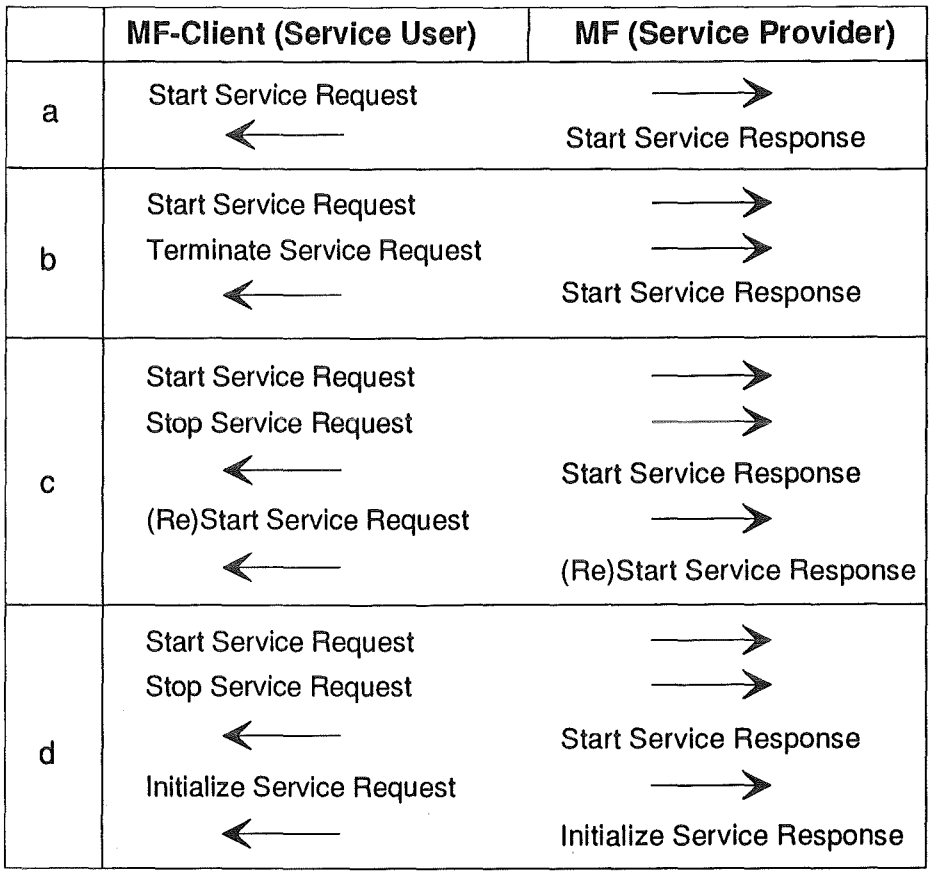


Fig. 6.8 Service Sequences of Operations

Considering that the ***Manufacturing Message Specification (MMS)*** is the most recommended standard for the manufacturing message transfers, it was conceived as being a good starting point for the ***Service Units*** to be constructed on the basis of the MMS service primitives. According to the needs for the test environment, several Service Units are specified. As stated in Section 4.2, a *Service Unit* is used to achieve a closed and consistent function. It may need several service primitives to achieve its function. For example, the ***Service Unit*** for loading a program requires three MMS service primitives, namely: *InitiateDownLoadSequence*, *DownLoad-Segment* and *TerminateDownLoadSequence*.

It should be pointed out that in this work it was not the intention to define a complete set of *Service Units*. More importantly, it is necessary to validate the proposed approach to the MF design and to investigate the behaviour of the Machine Front-End in a CIM-OSA System. Some *Service Units* defined for the case study are given in [*Hou93b*].

7) Implementation and Validation

This chapter outlines the creation and testing of a MF prototype. The prototype was implemented in the C-programming language using the approach proposed in the previous chapters. A testing environment has been defined according to the available resources. Three CIM-OSA models have been used for the validation of the Machine Front-End, and the model described in Chapter 1 is discussed. Finally, the results of the validation are evaluated. The test scenario was presented for the ESPRIT-AMICE Project as the first CIM-OSA demonstrator and has been registered as the McCIM System, a KfK development of the CIM-OSA system.

7.1) Goals of Validation

Verification and validation are important activities in the overall software life cycle. ***Verification*** is defined to be the comparision at each stage of the software life cycle to determine that there has been a faithful translation of that stage into the next one. ***Validation*** is to determine the level of conformance between the system requirements and an operational sofware system under operational conditions [*EWIC83-84*].

Validation (***system testing***) is undertaken when all the related hardware and software have been integrated and the whole system has been established. It will assure that the code meets the requirements expressed in the software requirements specification. A validated system can still exhibit unsatisfactory performance, however, due to poor, incomplete or erroneous specifications [*Quir85*].

A guideline for the verification and validation of critical computer systems and some applicable techniques are discussed in [*Redm88*]. System validation should detect software errors and estimate the overall system reliability based on the results of systematic testing, probabilistic testing and operating experience.

Within this dissertation, a more realistic validation of the Machine Front-End was considered. It was not possible to accomplish a full testing of the Machine Front-End, under all possible operational conditions, because of the inavailability of certain resources. However, the validation could show with the available resources, how far the proposed approach can achieve the two main objectives given in Section 2.2, the

conformance with the CIM-OSA concept and the satisfication of the industrial user requirements.

To enable this validation a MF prototype as well as a testing environment were established. Both of them are described in the following sections and the evaluation of the results is discussed in Section 7.6.

7.2) Prototyping of the Machine Front-End

For the prototyping of the Machine Front-End (MF) an implementation, instead of simulation, was chosen. The validation by means of an implementation needs more effort than simulation, but it provides an opportunity to justify the concept. Moreover, it gives a possibility to closely examine the specifications and to perform the stepwise refinement. This choice was identical with the main goal of the ESPRIT-AMICE project 5288.

Figure 7.1 shows the prototype of the MF Control Engine which was implemented in the C-programming language, the most portable language available now. The implementation uses the modular design technique which can easily be realized by the C-programming language.

In the figure, circles indicate information, while boxes are for the procedural operations. The *Initialize Service Module*, *MFInitMod()*, selects the Control Models required for running the session. It generates control structures for the access to both the *Control Model Library* and the *Abstract Objects.* The '*Loaded Control Models*' and '*State-Transition Diagrams of Abstract Objects*' are static, i.e. they stay in the working memory and will not be modified.

The *Receiving Service Module*, *RecvServMod()*, gets messages which are queued in the Communication Services and are waiting for the Machine Front-End. There are two types of input messages: the MF-Service Requests (MFO Call) issued by the MF-Clients and the responses from the MF-Servers. The MF-Service Requests are put into the *Pending Indication Queue*, while the responses are stored in the *Pending Request Queue.*

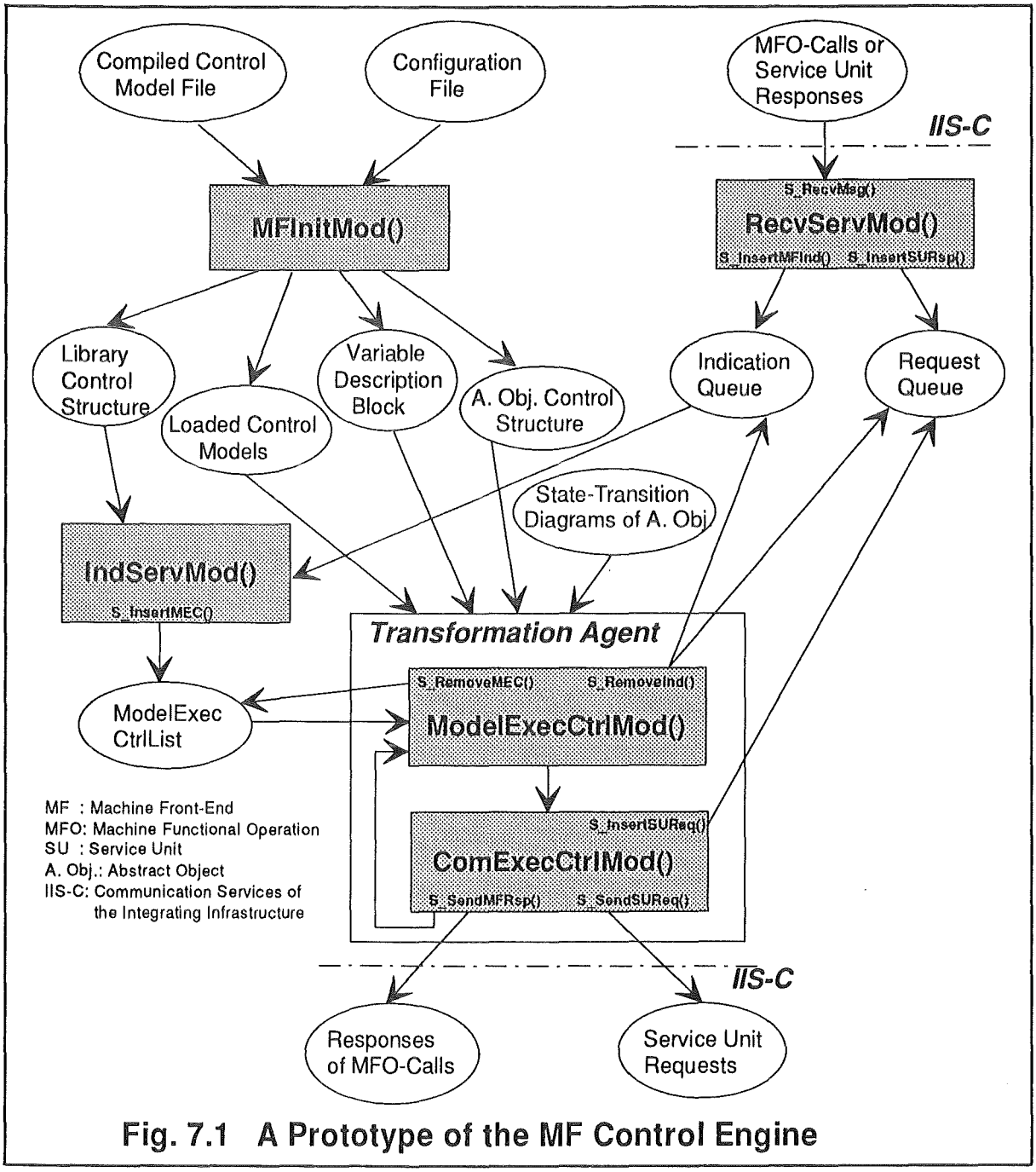


Fig. 7.1 A Prototype of the MF Control Engine

The '*ModelExecCtrlList*' contains a list of control records. Each control record holds information for the processing of a Control Model for a specified MFO execution. These control records are initiated by the *Indication Service Module, IndServMod()*, and used by the *Transformation Agent*. Within the processing of a Control Model the Transformation Agent may send requests to the underlying *Service Unit Provider* for controlling the MFO execution at the MF-Server. As soon as a Control Model is

completely processed, the Transformation Agent sends a response back to the MF-Client.

Some aspects described below have been regarded in this MF prototype. These aspects are important for the development of a complete Integrating Infrastructure as well as for its portability. These are:

- Clear definition of interface functions: For sending and receiving messages they are realized by three interface functions, S_RecvMsg(), S_SendSUReq() and S_SendMFRsp(). Only these three functions are dependent on the environment which the Machine Front-End is connected to.
- Access to information by use of a separate control procedure: The Machine Front-End uses a control structure to access the loaded Control Models. Similarly, it uses a control structure to access the description of *Abstract Objects*. This concept allows to add or remove 'static' information (Control Models or *Abstract Objects*) without changing the control mechanism. Thus, it allows the easy insertion of the further developed *Abstract Objects*, it needs just the insertion of the 'static' description of *Abstract Objects* without changing the control procedure.
- Portability of the MF program: The implementation in the C-programming language makes the MF program easily portable to other operating systems.

7.3) Requirements for Testing Environment

The Machine Front-End is a component of the Integrating Infrastructure (IIS). It, together with other IIS components, is to achieve the execution of CIM-OSA models resulting from the Modelling Framework. ***Ideally***, the validation of the proposed approach of the Machine Front-End requires:

- a complete implementation of the Integrating Infrastructure;
- several well-designed CIM-OSA models, resulting from the Modelling Framework, which describe the real manufacturing activities;
- a real environment with appropriate software and manufacturing devices;

To establish such a scenario was beyond the scope of this dissertation, since too many components of CIM-OSA still exist in the conceptual phase. The Machine

Front-End, proposed and developed in this work, is the first contribution to CIM-OSA, which has resulted in a prototype. None of the other IIS components is available.

Nevertheless, the testing environment for the validation of the Machine Front-End (MF) should contribute to achieve the following goals:

- investigation of the MF capabilities and behaviour in a CIM-OSA system;
- demonstration of the '*Executable Model*' which is the key purpose of the Integrating Infrastructure;
- establishing of a basis for integrating the successive implementation of other IIS components.

Considering available software and hardware, ***the resulting requirements*** for the testing environment were given as follows:

- Implementation of the Integrating Infrastructure: The Business Services should be implemented to some extent to achieve its functional flow, and the Communication and Information Services need only to be implemented to make the test of the MF prototype possible.
- Service Unit Provider: It should preferably use a software product which implements the international standard MMS.
- Network Communication: It should preferably use the international standard for the manufacturing communication MAP (Manufacturing Automation Protocol). There were three MAP-Boards and the communication protocol software from CONCORD Co. available [*CONC90a-b*].
- Manufacturing Devices: There were two robots of type 'Mitsubishi RV-M1' available [*Mits90*];
- Domain Process Model: It should be designed such that it is feasible with the resources available and the realization time. However, some MF capabilities, e.g. parallel processing, monitoring and synchronization, should be able to be verified by use of the model.
- Computer Stations: There were several IBM compatible personal computers available;
- Operating System: It should be a multi-tasking operating system which is discussed below;

A CIM-OSA system is a distributed and concurrent system, so that the Integrating Infrastructure (IIS) must co-operate with a number of application processes. These application processes can be installed on various network nodes and can be run on different operating systems. The Integrating Infrastructure itself also needs several processes for its implementation which can be also installed on several network nodes. Therefore, a CIM-OSA system requires a multi-tasking operating system.

According to the CIM-OSA concepts, the Business, Front-End and Information Services use the callable functions of the *System Wide Exchange* (an element of the Communication Services) to exchange messages. Multi-tasking operating systems usually support interface functions for message passing between processes. Therefore, from the viewpoint of practical implementation the callable functions of the *System Wide Exchange* should be realized by using the interface functions supported by the applied operating system. This in turn means that the implementation of the *System Wide Exchange* will be strongly influenced by the operating system used, while the implementation of other IIS components is independent of it. In conclusion, the following ***restrictions*** on the testing environment were identified:

- The choice of an operating system was limited by the computers used and the software products available. Considering these two points, the MicroSoft Windows operating system was chosen;
- The data exchange between the IIS processes directly used the *Dynamic Data Exchange* supported by the MicroSoft Windows, and the transparency of access and location was not concerned in this work. The reason was that the specification of the Communication Services was not yet stable and still needed improvement by AMICE;
- The Information Services was implemented so far just for the storage and retrieval of the process data. No functions specified by AMICE and data conversions were incorperated;
- The Business Services was simulated in functional flow by use of the Petri-Nets Tool, PACE [*PACE91*]. PACE supports utilities for the creation of CIM-OSA models and the graphical execution of the created models down to the level of Functional Operations;

- The Service Unit Provider used the LiveData [*CYCL92*], an implementation of MMS, because of its ease of use and a mutual cooperation with Cycle Software Inc. This product is still a ß-version (testing by user).

The testing environment as well as a partial implementation of the Integrating Infrastructure are described in the next section.

7.4) Testing Environment

Figure 7.2 gives the overview of the testing environment. It consists of three stations connected by a MAP network: one station acts as a cell controller on which the IIS processes are installed; the other two as MF-Servers containing machine control programs. The testing environment with detailed description of system components is depicted in Figure 7.4.

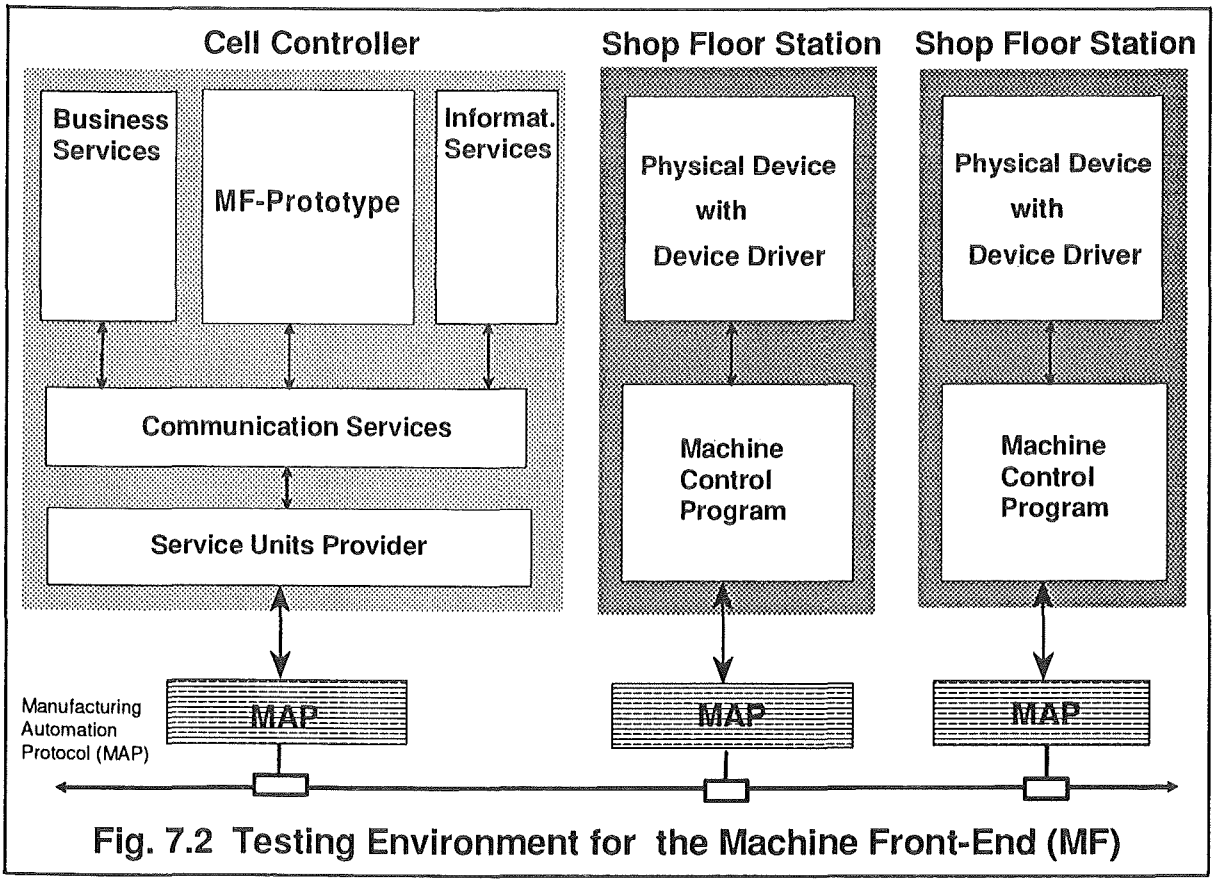


Fig. 7.2 Testing Environment for the Machine Front-End (MF)

Figure 7.3 describes the IIS processes and the message exchange between them. In this prototype, the ***Dynamic Data Exchange*** of *Microsoft Windows* [*Micr90*], instead of the *System Wide Exchange* of CIM-OSA, for the message exchange between the IIS processes was used. Three message types of *Dynamic Data Exchange* are shown in the figure: POKE is used by a client process to send a message to the server process without waiting for the attention by the server process; Both REQUEST and DATA build a synchronous dialogue to get a message from a server process. The numbers in circles help in understanding the sequence of a MFO execution.

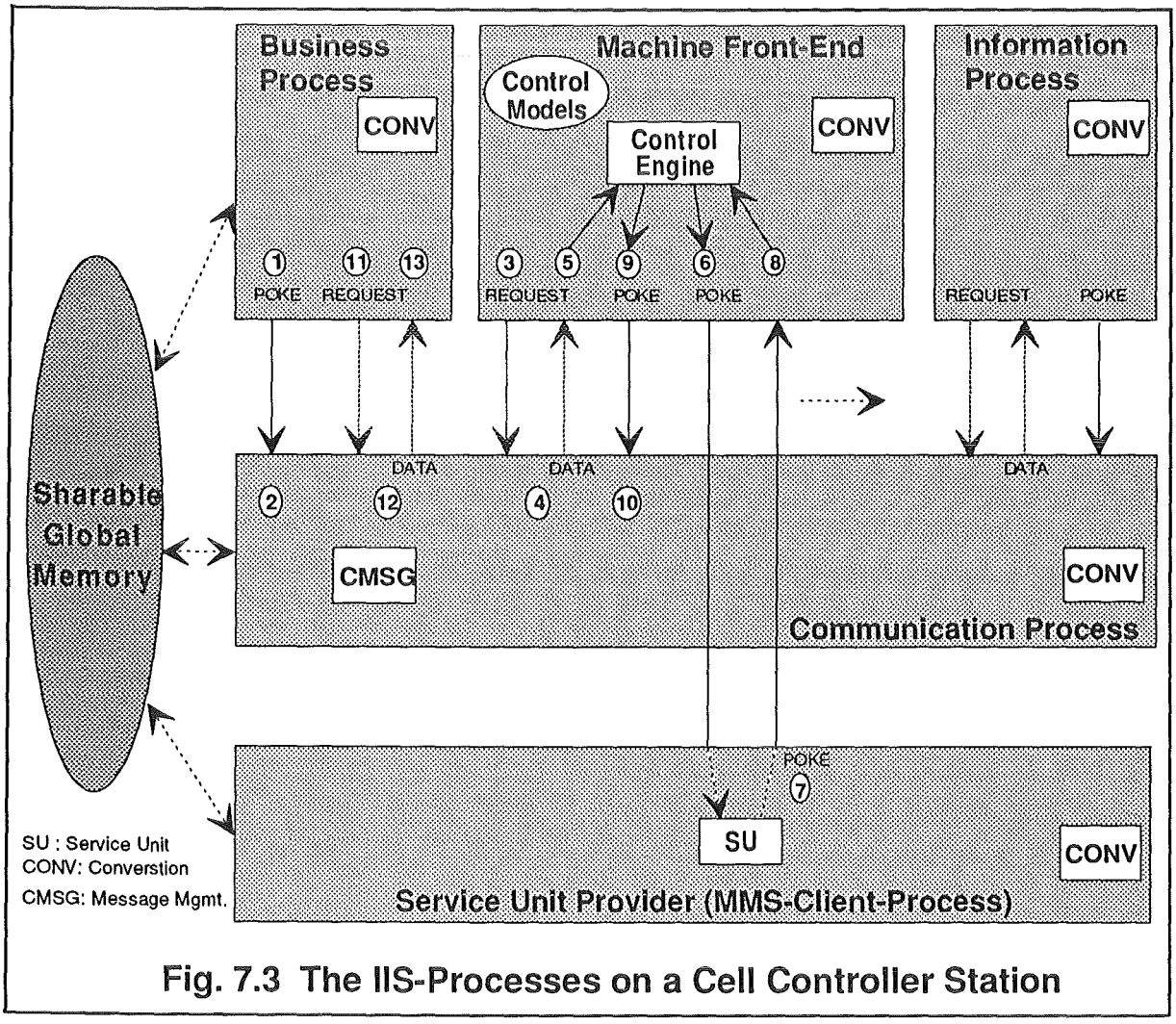


Fig. 7.3 The IIS-Processes on a Cell Controller Station

The prototype uses the concept of sharable global memory to store the messages to be transferred between the processes. Each message unit has a unique identifier assigned to it. For the message passing, the method of *message identifier transfer* is

applied. The message exchange between the Business, the Machine Front-End and the Information Process (via the Communication Process) uses the *MF-Access Protocol Data Unit*, while between the Machine Front-End and the Service Unit Provider the *MF-External Protocol Data Unit* is applied. Both types of *Protocol Data Units* are described in Section 6.4.

Each process has a *Conversation Module* (*CONV*) which is responsible for the management of conversations (associations) with other processes. The Communication Process includes a *Message Management Module* (*CMSG*) which manages the messages queued in the Communication Process. In the case of a synchronous dialogue, established by REQUEST and DATA, a time-out for the duration of the request is set to prevent the dead-lock situation where the client process is waiting for a response from a server process which is not able to respond.

The whole IIS Processes, including the MF prototype, were implemented in C-programming language by the author, while the other system components were realized by several colleagues.

Figure 7.4 depicts the system components of the testing environment which includes two demo applications: one for a ***Casting Plant*** and the other one for an ***Automobile Plant*** from the two industrial partners of the ESPRIT-VOICE project. These two applications are shown inside dotted lines, respectively [*VOIC93*]. The Business Process has been replaced by an Interface and a PACE Station. The Interface, on one side transfers the data between the PACE Station and the Cell Controller Station via a serial port of RS 232, and on the other side it sends and receives data from/to the Communication Services. The PACE Station uses the Petri-Nets tool PACE to create the CIM-OSA models via a graphical interface and to execute the model down to the level of Machine Functional Operations [*DiNe91*,*DNBK93*].

In these two demo applications, several packages for the process simulation and visualization, such as InTouch, Excel, and VisualBasic have been added. The InTouch package provided the simulation and animation of the aluminium process of the Casting Plant [*InTo92, Gyme93*]. The Excel package produced the graphical representation of the product and process tracking data [*Exce90, Rafi92*]. The VisualBasic package was used for the operator interface, the robot process visualisation, and the robot interface [*Visu91, Guit92*]. The operator interface was incorperated to input the setup points in the Casting Plant Application, as well as to

provide a control panel which allows the Automobile Plant Application to be terminated, stopped and restarted. The robot interface between the MMS-Sever program and the robot devices was required because the LiveData software package does not support a connection to the serial port of RS232.

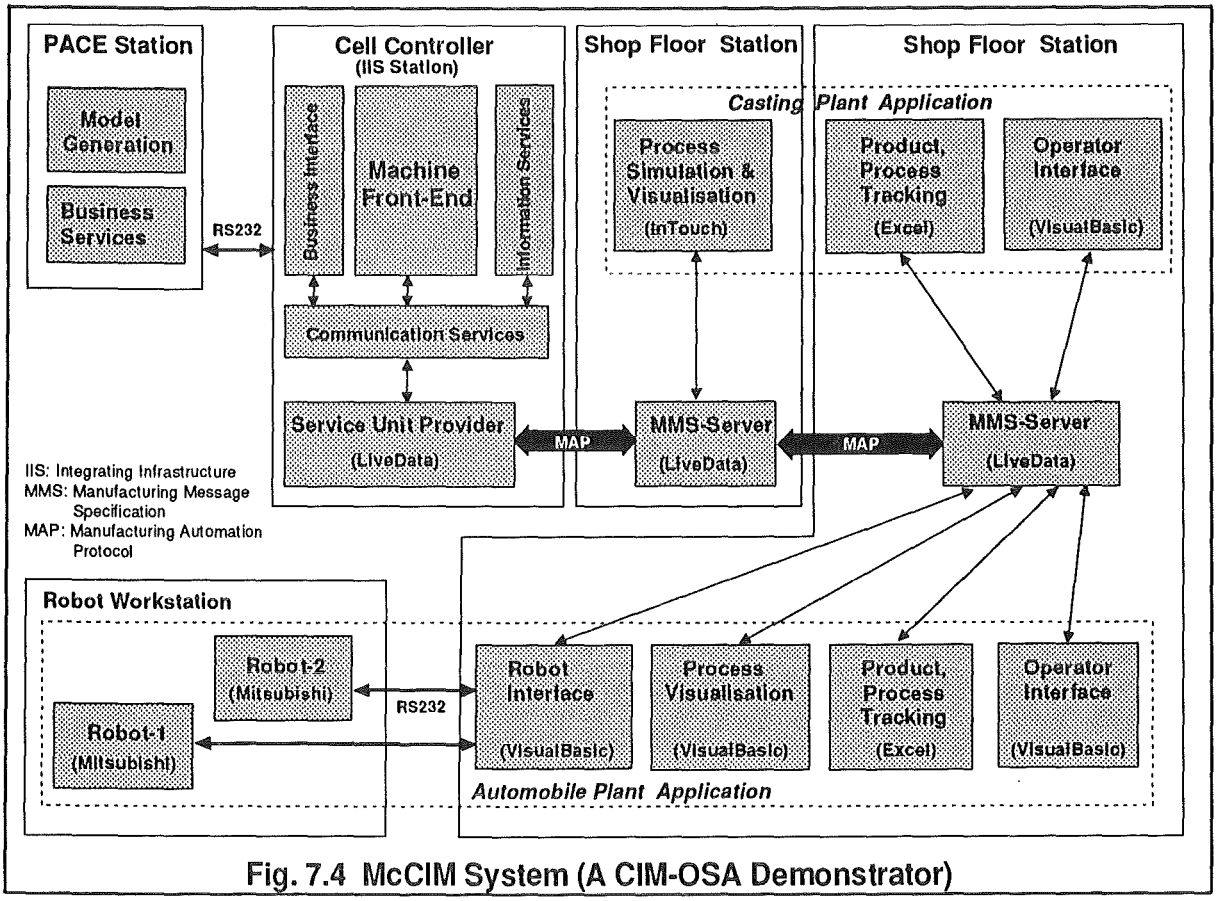


Fig. 7.4 McCIM System (A CIM-OSA Demonstrator)

This test scenario has been registered as the McCIM System, a KfK development of CIM-OSA system. Based on this work as well as the experience gained, it is the intention to launch further CIM projects.

7.5) Case Study

Three CIM-OSA models (Domain Process Models) have been designed as input for the validation of the Machine Front-End. The models were designed as close as possible to meet the real manufacturing activities, but had to be a compromise with respect to feasibility (w.r.t. the resources available) and realization time.

Two viewpoints of the Domain Process Model, which was designed by the author and introduced in Figure 1.10, are discussed below:

- the Procedural Rules and Functional Operations of the model;
- the Control Model for a Machine Functional Operation.

The other two Domain Process Models, which were based on the model above, can be found in [*VOIC93*]. They were designed particularly to meet the requirements of the two industrial project partners of ESPRIT-VOICE. Actually, these three models have a similar structure, but the model in Figure 1.10 gives a clearer understanding about the view points above without having to describe the requirements of the respective industrial project partners.

However, it should be also pointed out that some aspects in all the three models - like information and resource management - are not considered. This is because that other IIS components are not completely implemented.

The Procedural Rules and Functional Operations of the model

To get a better understanding of the description below, it is recommended to recall the description about the processing of a Domain Process Model by the Business Services described in pages 22-23.

In Figure 1.10, suppose that PR1 contains two logical objects and PR2 only one logical object. They are initialized with the value FALSE. When both robots are successfully set up, the two logical objects of PR1 will be set to TRUE. The Activity Control can then simultaneously start the occurrences of EA2, EA3 and EA4 on the requests of Business Process Control. When an operator terminates the scenario, both the EA3 and EA4 occurrences will be terminated first and consequently EA2 will terminate itself. After the termination of the EA2 occurrence the logical object of PR2 is set to TRUE and the Business Process Control will then request Activity Control to start the EA5 occurrence to close shop floor.

In order to reactivate the support for operator intervention, the HFO-20 should be restarted again every time the Activity Control receives a response from the operator. The HFO-20 will be terminated when both MFO-30 and MFO-40 are terminated. In this work, the MF Control Engine was used also for the control of the

HFO execution because the Human Front-End was not yet available. The functionality of the five FO's used in the model are given below:

a) Set-up: This type of MFO is used to download the required robot programs and to set-up the robot to an initial state.

b) Operator Intervention: This type of HFO supports an operator with a Control Panel which has three options (Stop, Restart and Terminate) to control the activities of the shop floor devices.

c) Control: This type of MFO is used to read the path coordinates of the master robot and to send them to the slave robot. With the received path coordinates the slave robot moves along its path accordingly.

d) Monitoring: This type of MFO is used to read path coordinates of both robots and put the deviations into a database. The data can then be displayed to an operator.

e) Shut-down: This type of MFO is used to reset the robot and upload the log files.

Since all the five types of FO's in the Domain Process Model apply to the same *MF_OC Abstract Object*, the Activity Control uses the same model of ***Standard Control Structure*** (cf. Section 6.4) to issue services to the Machine Front-End.

The Control Model for a Machine Functional Operation

As an example, the Control Model for the MFO-30 'Control', where three Service Units are used, is given below:

- *SU_ReadCoord* for reading the actual coordinate data of the master robot;
- *SU_WriteCoord* for sending the coordinate data to the slave robot;
- *SU_WriteSignal* for reporting the operation status of the Machine Functional Operation (MFO) on a Signal Panel.

The last Service Unit is introduced here to control the Signal Panel indicating the status of the MFO execution. Imagine that each status corresponds to a respective indicator. The *SU_WriteSignal* controls the appropriate indicator.

Since the Machine Functional Operation applies the *Abstract Object of MF_OC*, the Control Model should include four service blocks: each of them stands for the Start, Stop, Terminate and Initialize Service, respectively. The work to be done in each service block depends on the description of the Machine Functional Operation.

Except for the Start service block, all the other service blocks have the same function of reporting the operation status of the Machine Functional Operation (MFO). There are two types of reponses, the Start Service Response and the Initialize Service Response, which are to be sent back to the MF-Client (i.e. Activity Control). The status data for both of the responses are set by the Control Engine, according to the situation of the MFO execution. The Control Model given below explains itself by the comments.

```
MODEL  M_Control(ServiceData: TServiceData)
VAR /*  variables definition  */
    ...
BEGIN
    BLOCK  START    /* Start service block */
        /* variables initialization*/
        ...
        /* set operation status signal to Start */
        Set value of WriteSignalReqVar to 1;
        /* send the status data to the Signal Panel */
        REQUEST  SU_WriteSignal_Id,  WriteSignalReqVar,  WriteSignalConfVar;
        WAIT          WriteSignalConfVar;
        /* read the first coordinate data of the master robot */
        REQUEST   SU_ReadCoord_Id,      ReadReqVar,      ReadConfVar;
        REPEAT
          /* wait until the coordinate data has arrived */
          WAIT          ReadConfVar;
          /* copy the coordinates of master robot to WriteReqVar */
          ...
          /* send the data to the slave station, the salve robot follows the path*/
          REQUEST   SU_WriteCoord_Id,     WriteReqVar,     WriteConfVar;
          /* read next coordinate data of the master robot */
          REQUEST   SU_ReadCoord_Id,      ReadReqVar,      ReadConfVar;
          /* wait until the moving of the slave robot has been finished */
          WAIT          WriteConfVar;
        UNTIL (TRUE);
    BLOCKEND;

    BLOCK  STOP     /* Stop service block */
        /* set operation status signal to Stop */
```

```
        Set value of WriteSignalReqVar to 2;
        /* send the status data to the Signal Panel */
        REQUEST  SU_WriteSignal_Id, WriteSignalReqVar, WriteSignalConfVar;
        WAIT        WriteSignalConfVar;
    BLOCKEND;

    BLOCK   TERMINATE     /* Terminate service block */
        /* set operation status signal to Terminate */
        Set value of WriteSignalReqVar to 3;
        /* send the status data to the Signal Panel */
        REQUEST  SU_WriteSignal_Id, WriteSignalReqVar, WriteSignalConfVar;
        WAIT        WriteSignalConfVar;
    BLOCKEND;

    BLOCK   INITIALIZE    /* Initialize service block */
        /* variables initialization */
        ...
        /* set operation status signal to Initialize */
        Set value of WriteSignalReqVar to 4;
        /* send the status data to the Signal Panel */
        REQUEST  SU_WriteSignal_Id, WriteSignalReqVar, WriteSignalConfVar;
        WAIT        WriteSignalConfVar;
    BLOCKEND;
END
```

7.6) Testing and Evaluation of the Results

On the validation of the proposed Machine Front-End, the two points below should be identified first:

- what is the goal of the validation?
- which attributes are important for the system evaluation?

The two goals of the validation in this work has been stated in Section 7.1, so that the validation has to prove how these goals can be achieved.

The first goal, ***the conformance with the CIM-OSA concepts***, has been achieved by means of:

- the IIS Client-Server Architecture which has been examined through the information exchanges within the Integrating Infrastructure as well as between the Machine Front-End and the machine controllers;
- the contribution to the 'Executable Model' which has been proved by successful use of the three CIM-OSA models in the Test Scenario;
- the portability of the implemented Functional Operations which has been achieved by the Control Models that are implemented independent of the manufacturing environment;
- the Integration of heterogeneous manufacturing devices which will be discussed together with the second goal below.

The second goal, ***the satisfication of the industrial user requirements***, has been validated through the observations by means of the C-Programming Debugger and the Testpoints-Display during the testing of the McCIM system with the three CIM-OSA models. The following will discuss ***some important attributes for the system evaluation*** *with regard to the industrial user requirements* listed in Section 2.2, which are:

- Monitoring and data collection: The process data captured for the monitoring needs an immediate display to the operator, while the data for the data collection is stored for analysis for later use. These have been achieved by the Machine Front-End of the McCIM system. The number of devices that can be monitored or served by the Machine Front-End are unlimited, but they will influence the processing speed required by the user;
- Control and synchronization: The Domian Process Models used for the case studies have included these two functions, and the success of the McCIM system has proved that they have been accomplished;
- Alarm and Emergency handling: The alarm handling may require the operator to readjust the system back into the normal state, while the emergency signal indicates that the system state is critical and could cause damage, therefore the operator should immediately take actions. The alarm handling has been proved through the support for the input of setpoint values in the *Casting Plant Application*, and the emergency handling through the support of a Control Panel in the other two applications. The reaction time required by the alarm and emergency handling, as well as the strategies for the emergency handling, will be discussed later in this section;

- Time critical operation: The proposed approach provides two mechanisms for the time-critical MFO's: the ***priority-driven operation*** and the ***control-oriented execution***. The MFO-calls with a higher priority are processed by the Machine Front-End prior to those with the lower priority, and the Machine Front-End can be asked to control the execution of a MFO which doesn't want to be interrupted until it is completed. These two mechanisms provide a possibility of improving the operations of the time-critical MFO's. The time required by the execution of a MFO will be also discussed later in this section;
- Integration of heterogeneous manufacturing devices: This can be realized by the proposed concept of Service Units. The Service Units are implemented in the IIS underlying process which involves the device-dependent services (cf. Fig. 6.5);
- Integration engineering tool: This concerns tools that can be used for the CIM-OSA modelling task, the configuration of the Integrating Infrastructure and the implementation of a CIM-OSA system. It belongs to one of the main tasks in ESPRIT-VOICE as well as AMICE projects, and is not dealt with in this work.

The attained MF capabilities have been matched with the user requirements and are represented in the table below.

In the table, cross (x) means that the particular capability is involved in satisfying the user requirement. The capabilities highlighted by the symbol of '*' denote that they are not directly related to the Machine Front-End and is a facility provided by the other CIM-OSA components which are not yet available. For example, the *Access Transparency* which is to hide the local/remote accesses is the task of the Communication Services, not of the Machine Front-End. These capabilities are only included here because they are needed to achieve the user requirements, identified in Section 2.2.

The fulfilment of a particular user requirement may need several MF capabilities; for instance, the 'Remote Process Monitoring' requires the Parallelism, Timing/Polling, REQUEST-/WAIT-Operation and Service Units. The MF capabilities (1-7) were described in detail in the previous chapters. The *Indirect Intervention* is described later in this section and the *Migration Transparency* is discussed in Chapter 8.

MF Capabilities / User Requirements	1) Parallelism	2) Timing / Polling	3) Priority-driven	4) Execution-oriented	5) REQUEST-Operation	6) WAIT-Operation	7) Service Units	8) Indirect Intervention	9) Access Transparency *	10) Migration Transparency	11) Command Language	12) Engineering Tool *
a) Remote Process Monitoring	X	X			X	X	X					
b) Data Collection	X	X			X	X	X					
c) Synchronization	X				X	X	X					
d) Alarm Handling					X	X	X	X				
e) Emergency Handling			X	X				X				
f) Time Critical Operation			X	X								
g) Local/Remote Access									X			
h) Integration of Vendors Devices							X			X		
i) Implementation of FOs											X	
j) Integration Engineering												X

Execution Time of a Machine Functional Operation (MFO)

From the viewpoint of *OSI-Reference Model*, all the IIS-processes are placed on top of the OSI-7th layer. Actually, the Integrating Infrastructure (IIS) itself has also a layered structure. There are three layers distinguished in the IIS implementation: the Service Unit Provider with *Application Service Elements* (e.g. MMS, FTAM) in the lowest layer, the Communication Process in the middle, and the Business, Front-End and Information Processes in the highest layer. It is conceived that through this multi-layered structure the execution of a Domain Process Model needs more time than a traditional program involved in the same task.

In addition to the above table which shows the evaluation in a qualitative way, the possibility to attain the performance of the Machine Front-End as a quantitative value was examined. Through the observation during the testing phase, it was concluded

that the evaluation of the MFO execution time can not identify the MF performance because the time required by the Machine Front-End itself is not significant in comparison with the time required by other contributing system components. The factors which dominantly influence the execution time of a MFO are:

- the functionalities of the MFO's to be achieved and the process time of manufacturing devices;
- number of the CIM-OSA models or MFO's to be processed concurrently;
- the multi-tasking operating system on which the IIS-processes are installed: The time required for the execution of some operations within a process is strongly impacted by the task switching mechanism of the applied operating system.
- the applied software packages and hardware.
- the underlying Communication Subsystem;

Figure 7.5 depicts the components used for proving the above statement. It was part of the testing environment shown in Figure 7.4.

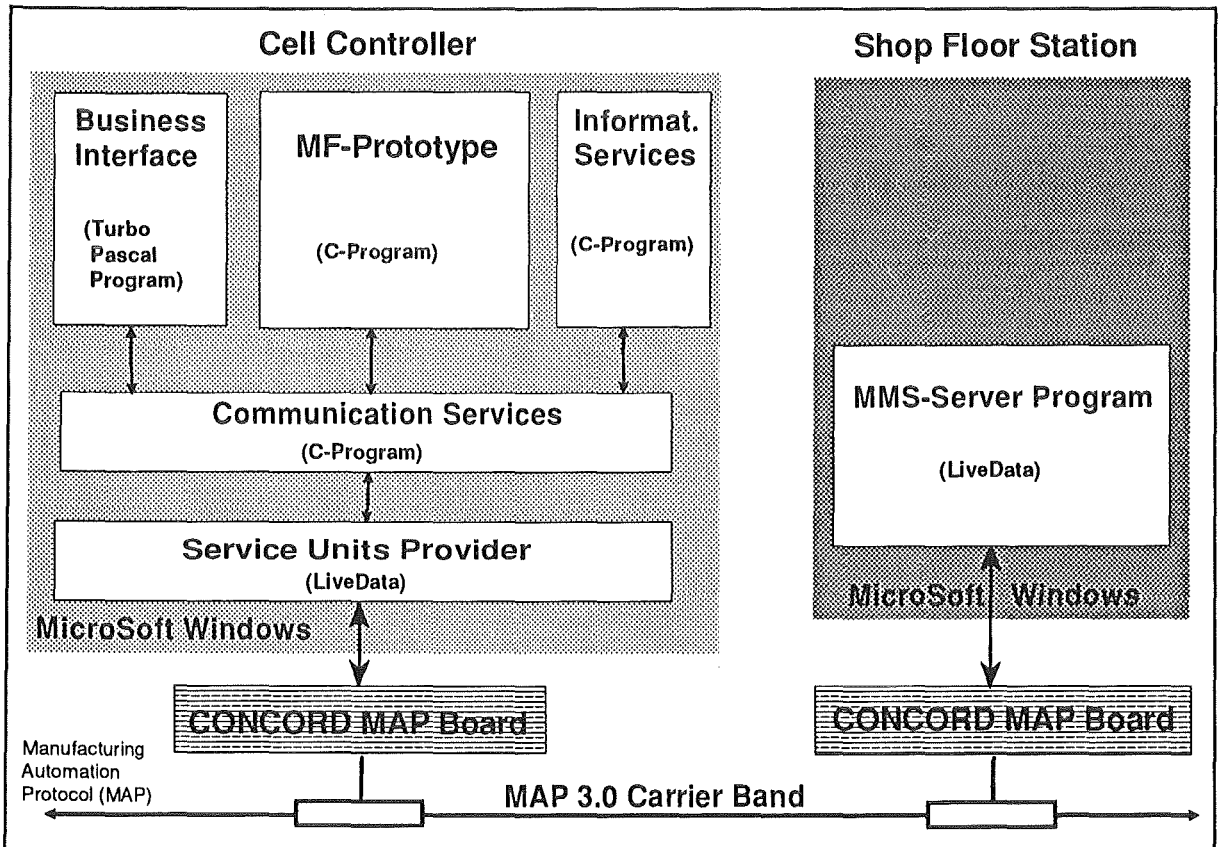


Fig. 7.5 Configuration for the Measurement of the MF Performace

The components were:

- the cell controller consisting of a Service Unit Provider and all the IIS processes, which was installed on a Toshiba T6400DX personal computer with clock frequency 33MHz;
- one shop floor station consisting of a MMS-Server Program, which was installed on an IBM compatible personal computer with 33MHz. No manufacturing device was connected;
- a MAP 3.0 carrier band based on CONCORD Boards;

In addition to get the performance of the Machine Front-End, the time required by the contributing system components for the execution of a MFO was also evaluated.

A set of timer functions with resolution 1µs was provided for the time measurement [*Kais89*]. They were inserted into the programs. For the input of the Machine Front-End, three types of MFO's were implemented. They were sent directly by the Business Interface. These three types of MFO's were:

a) a MFO which required only one Service Unit for writing a value to the shop floor station;

b) a MFO which required only one Service Unit for reading a status data from the shop floor station;

c) a MFO which required the above two Service Units.

The Business Interface sent the next MFO only when the previous MFO had been completed. This means that the Machine Front-End had only one MFO to process at a time. In each of the three experiments, 100 of the same type of MFO's were used for the measurement. Since the three experiments have similar results, only the first type of MFOs' result is shown in the following figures 7.6 and 7.7. Figure 7.6 depicts the process time of a MFO required by the contributing system components. The curve numbers indicate:

1 = ***MF Process Time*** which is the time required by the Machine Front-End for the queue management, access of the Control Model Library and processing of the corresponding Control Model. The curve of the MF Process Time remains very close to the X-axis, and so is represented by the circles;

2 = ***Task Switching Time*** which is the time required for the task switching operations by the MicroSoft Windows Operating System. It includes only the task switching time within the MF Process;

3 = ***Service Unit Execution Time*** which is the time required for the execution of a Service Unit including the underlying communication subsystem and the shop floor station;

4 = ***MF Total Time*** which is the time required for the processing of a MFO by the MF Process. It is calculated from the receiving of a MFO-call to the sending of the response, and is the sum of the *MF Process Time*, *Service Unit Execution Time* and *Task Switching Time*;

5 = ***MFO Execution time*** which is the time required from the issue of a MFO-call by the Business Interface to the arrival of the response. It includes the *MF Total Time* and the time required by the other IIS processes.

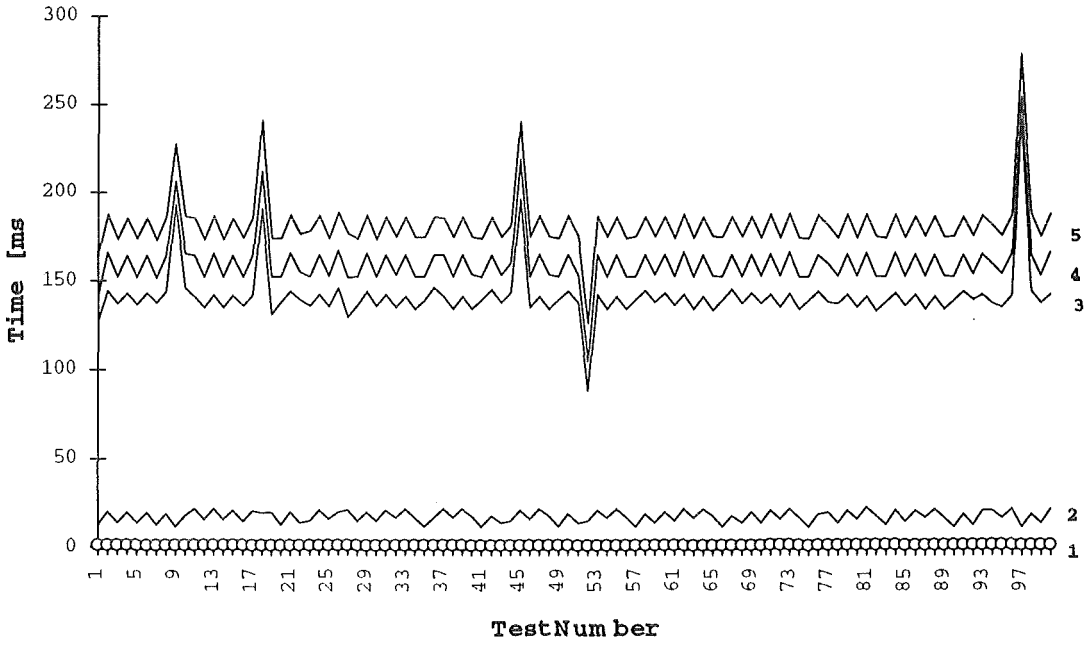


Fig. 7.6 Execution Times with Single MFO

The time required by the Machine Front-End itself, as predicted, was not significant, and takes only an average of 1.3ms (i.e. 0.71% of the *MF Execution Time*). The *Task Switching Time* takes 17,7ms (i.e. 9,7%) with a standard deviation 3.4ms. The *Service Unit Execution Time* takes a large part of the required time; an average of 141,6ms (i.e. 77,6%). It will increase when the manufacturing device is connected, e.g. a Service Unit for the movement of a robot arm from one point to another point

can take several seconds. The *MF Total Time* and the *MFO Execution Time* include the *Service Unit Execution Time*, so the three have a similar curve. Some peaks appeared in the *Service Unit Execution Time*, which were assumed to be caused by the Service Unit Provider (implemented in LiveData) and the MicroSoft Windows.

Figures 7.7 shows the frequency distribution of the *MF Process Time*. It was hard to recognize the kind of distribution because different curves appeared in several experiments with the same type of MFO, even if they had the same average time.

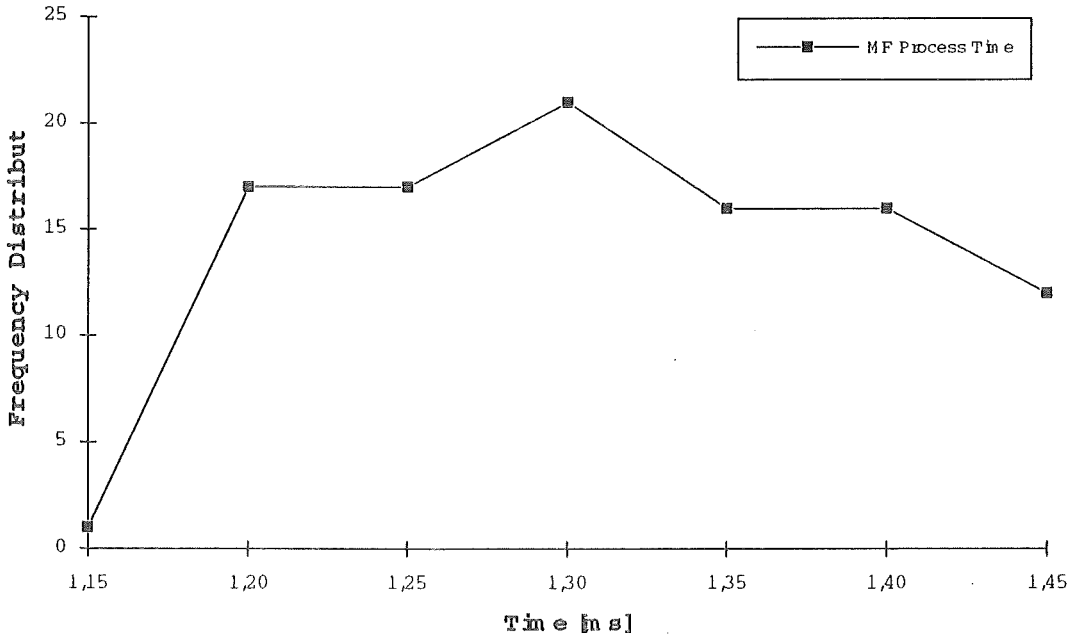


Fig. 7.7 Frequency Distribution of the MF Process Time

Another evaluation was performed for the investigation of the system behaviour on the processing of multiple MFO's concurrently by the Machine Front-End.

Figure 7.8 shows the MFO Execution Time of three MFO's with the same first type, and Figure 7.9 depicts the time interval between two Service Unit Requests (SU-Calls). The three MFO's were processed concurrently by the Machine Front-End. Similarly to the result shown in Figure 7.6, most of *MFO Execution Time* was for the execution of the Service Unit with remote shop floor station. The difference between two MFO Execution Times was about 200ms, even if the Machine Front-End sent the three Service Unit Requests to the Service Unit Provider in an interval of 23.8ms shown in Figure 7.9. This was also assumed to be caused by the Service Unit Provider and the MicroSoft Windows Operating System.

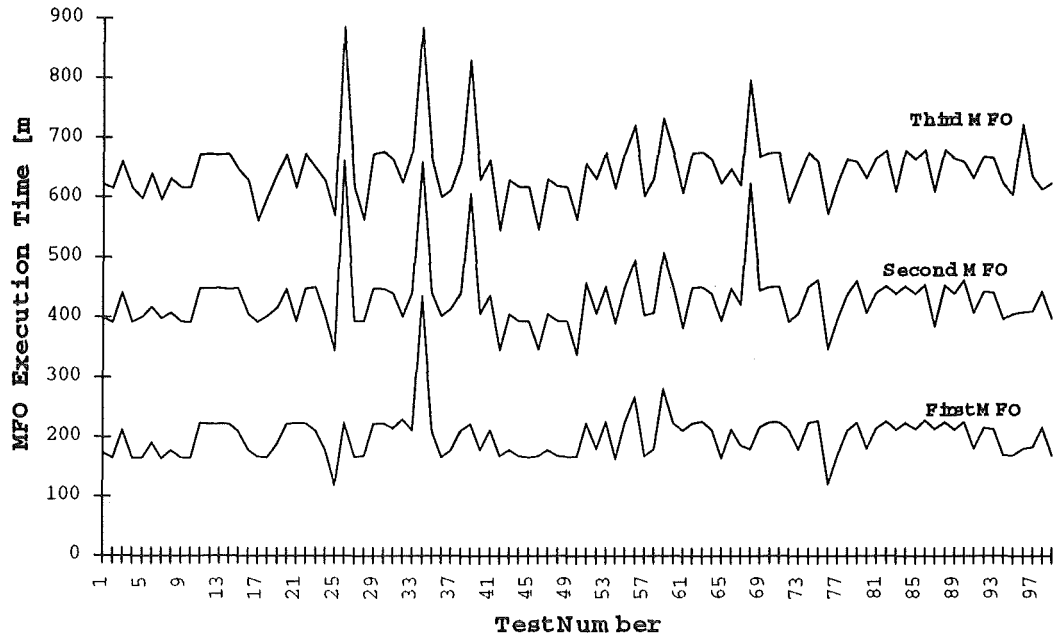


Fig. 7.8 The MFO Execution Time with three MFO's

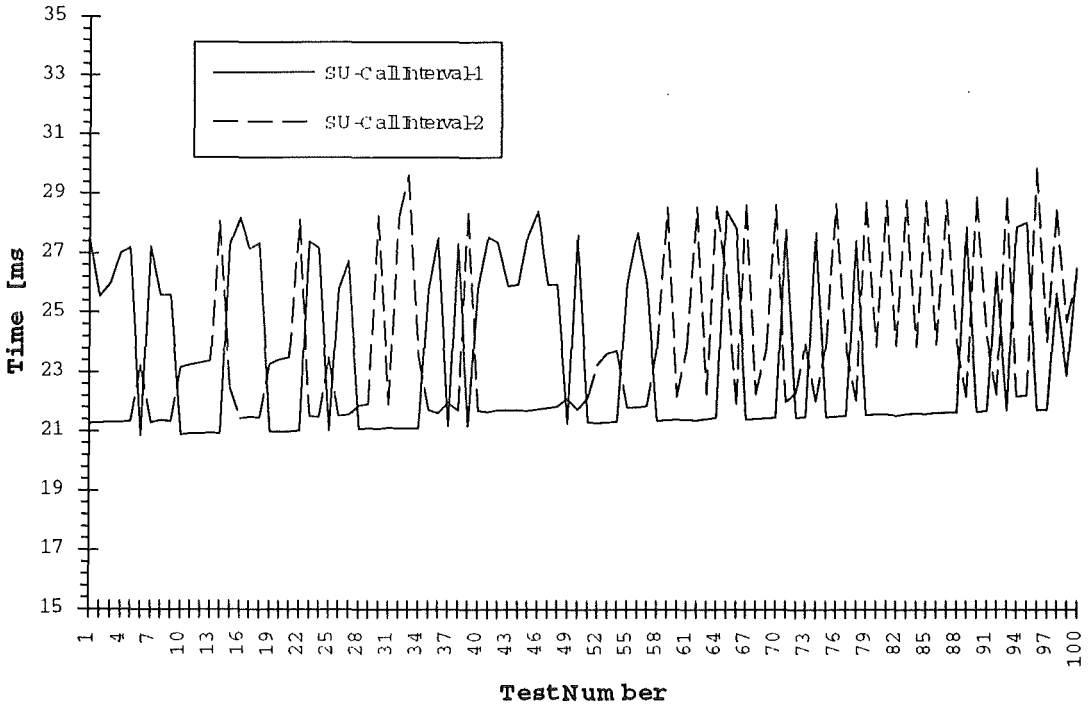


Fig. 7.9 Interval of the Service Unit Calls (with 3 MFO's)

A detailed analysis of these two packages using the methods described in [BoAk82] was not possible because of insufficient information. There are several methods discussed for the analysis of the performance of a distributed system, which include the analytical model building, simulation by use of the Queueing Network Models. The evalution of the Machine Front-End in this work was performed by use of the measurement method.

Figure 7.10 shows the result of 5 types of execution time by a number of MFO's. The X-axis indicates the number of MFO's being processed by the Machine Front-End concurrently. The Y-axis shows the average of total test runs for each type of execution time. The *MF Process Time* is still not significant. The *Task Switching Time* is almost constant, while the *Service Unit Execution Time* increases with the number of MFO's. The *MF Total Time* and the *MFO Execution Time* include the *Service Unit Execution Time*, so they have a similar curve. However, the *MFO Execution Time* increases marginally faster than the *Service Unit Execution Time* because it includes also the task switching time of the other IIS processes which increases also with the number of MFO's in process.

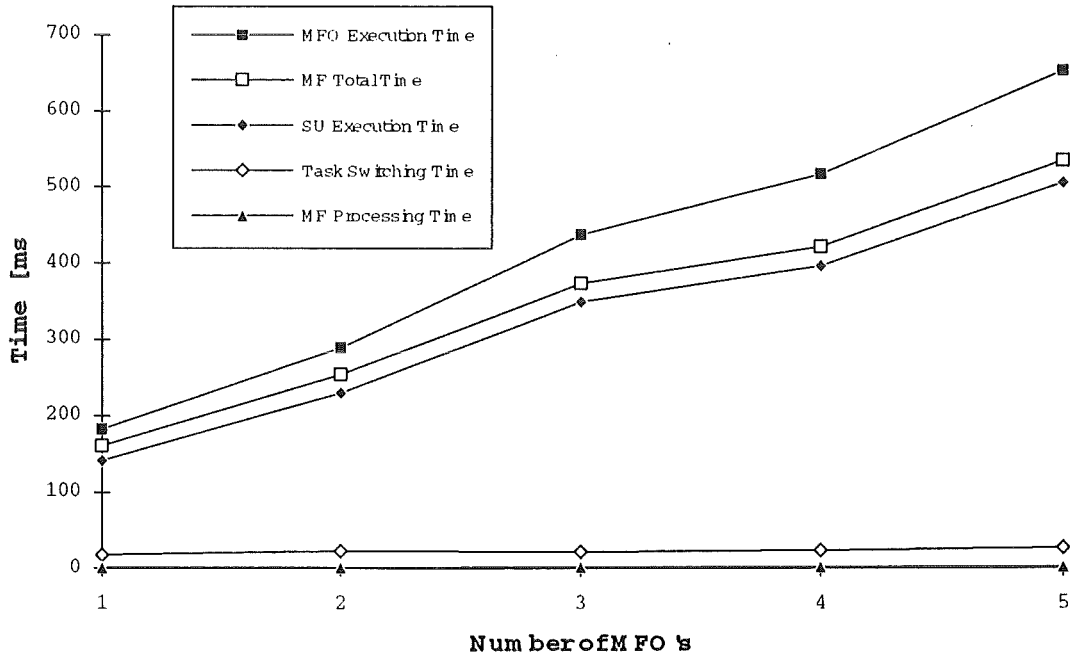


Fig. 7.10 Execution Times with Multiple MFO's

The table below gives some statistical values. They are presented as an average, for example with three MFO's being processed in parallel, each MFO Execution Time of the three MFO's was averaged first by 100 test runs, and then the mean of the three averages was calculated.

Time (ms)	MF Process Time	Task Switching Time	Service Unit Exec. Time	MF Total Time	MFO Exec. Time
1 MFO					
Average	1.292	17.660	141.588	160.540	182.436
Minimum	1.149	11.797	88.529	104.902	126.394
Maximum	1.441	23.073	241.199	254.474	278.633
Stadard Dev.	0.083	3.376	15.225	15.656	15.994
Percent (%)	0.71	9.68	77.61	88.00	100
2 MFO's					
Average	1.724	22.657	229.816	254.197	289.332
Minimum	1.394	19.111	170.704	195.170	230.515
Maximum	2.707	24.986	383.455	406.329	443.826
Stadard Dev.	0.260	1.698	27.574	28.023	29.122
Percent (%)	0.60	7.83	79.43	87.86	100
3 MFO's					
Average	1.972	22.617	337.159	361.748	425.250
Minimum	1.405	4.587	210.771	233.568	344.219
Maximum	2.842	26.118	571.441	596.794	659.598
Stadard Dev.	0.342	2.331	55.679	56.103	45.905
Percent (%)	0.46	5.32	79.29	85.07	100
4 MFO's					
Average	2.030	23.661	396.175	421.866	517.078
Minimum	1.541	7.522	308.093	338.240	443.044
Maximum	2.585	30.045	526.971	553.202	615.424
Stadard Dev.	0.286	3.292	41.987	41.533	32.229
Percent (%)	0.39	4.58	76.62	81.59	100
5 MFO's					
Average	2.174	27.339	504.548	534.061	651.962
Minimum	1.647	2.149	416.047	443.680	567.114
Maximum	2.794	65.879	762.153	785.748	863.850
Stadard Dev.	0.313	10.751	48.818	46.957	38.767
Percent (%)	0.33	4.19	77.39	81.92	100

Consequently, the evaluation results have shown that the time required for the execution of a MFO is influenced by the task switching mechanism of the applied operating system. The time required by the IIS processes themselves can be negligible in comparison with the other contributing system components. Therefore,

the execution time of a specified MFO can be estimated by the sum of the total task switching time and the total execution time of Service Units used within the MFO.

Strategies for the Emergency handling

In a CIM-OSA system, every activity in the external CIM-Modules is triggered and controlled by the Integrating Infrastructure by use of Domain Process Models. This implies that there should exist at least a *Human Functional Operation* in a Domain Process Model for the emergency handling, e.g. to stop a process by an operator. The *Human Functional Operation* should present to an operator a list of emergency handling methods and the *Human Front-End* should be able to accept the action given by the operator. To accomplish the operator action the following two strategies are considered.

- ***Direct intervention***: This strategy relates to the immediate interaction between the Human Front-End and the Machine Front-End. After the Human Front-End receives an emergency action from an operator, it sends a corresponding message to the Machine Front-End to change the control of the MFO execution directly;
- ***Indirect intervention***: This strategy relates to the interaction between the Human Front-End and the Activity Control to change the control of the MFO execution. The Human Front-End first sends a corresponding message to the Activity Control to change the value of control input objects (Procedural Rule) of the corresponding *MFO-Standard Control Structure*. This causes then the Activity Control to send a service request, e.g the Stop or Terminate Service Request, to the Machine Front-End to change the control of the MFO execution.

The first strategy immediately intervenes with a MFO execution. It can be used when the process requires a high degree of 'emergency' handling, but it needs a definition of the co-operation between the Machine Front-End and the Human Front-End. The second strategy causes a time delay because the operator action has to be passed first to the Activity Control, this gives a possibility that an operator can change not only a specified MFO execution but also the sequence within a Domain Process Model. In the case study, the second strategy is applied for the operator intervention and the Human Front-End uses the *MF Control Engine*.

The testing has proved that the proposed solution for the MF design conforms to the CIM-OSA concept, particulary in view of the Client-Server Architecture and the CIM-OSA Executable Model. Also the testing has proved that most of the industrial user requirements can be satisfied.

8) Recommendations with respect to CIM-OSA

This chapter discusses some important aspects for further developments in CIM-OSA. It concentrates on the CIM-OSA executable models, the generic approach to all three types of *Front-End Services*, and the migration of the proprietary machine control systems into the CIM-OSA system.

CIM-OSA is a long running research project of *the Commision of the European Communities* in the CIME area (*Computer-Integrated Manufacturing and Engineering*). It supports the efficient design, manufacture and distribution of high quality products, and provides European enterprises with the flexibility to respond rapidly to market opportunities. CIM-OSA applies the most advanced information technologies to: process modelling, real-time planning and scheduling, information integration, factory communication, automation, etc. Furthermore, CIM-OSA is an open system architecture which enables component developers, suppliers and end users to join together in product development.

However, CIM-OSA is still under development. Even if the main framework of the CIM-OSA concepts has been established, there are still details to be improved. These will be a challenge for research institutions, software houses, industrial CIM-users, etc.

Executable Model of CIM-OSA

CIM-OSA provides an integrated methodology from the process design to the manufacturing. A set of Integrated Enterprise Engineering Tools will be used to support an enterprise modeller to manipulate the constructs. The outcome of this task are the Domain Process Models with associated data objects. They are stored together in external Data Base Management Systems. The models and the data objects can be directly accessed and executed under the control of the Integrating Infrastructure (IIS). This feature enables a consistent and complete information processing from the process design to the manufacturing, and is known as a ***CIM-OSA Executable Model***.

The test scenario has shown the feature of a *CIM-OSA Executable Model* but only limited to the *Function View* of the modelling task. It would be interesting to examine the complete IIS behaviour by extending the case study to the *Information*, *Resource*

and *Organization View*, when the whole IIS prototype is available in the future. However, we have to keep in mind that the *CIM-OSA Executable Model* can be achieved only if the representation format of the process models and information objects from the *Modelling Framework* are clearly defined and standardized, so that the Integrating Infrastructure can understand and execute them.

Generic approach to all three types of Front-End Services

Actually, all three elements of *Front-End Services* have the same nature in the control of the FO execution, although the functionalities of the respective FO's remain different. The approach to the MF design separates the generic control mechanism from the FO-specific control knowledge. This approach can be adapted to the development of the other two elements of *Front-End Services*: *Application* and *Human Front-End*. In fact, this was shown in the test scenario where the execution of the Human Functional Operation is controlled by the Machine Front-End. The *MF Control Engine* is able to deal with all types of CIM-Modules implemented according to the CIM-OSA concept. These types of CIM-Modules are called ***CIM-OSA conformed CIM-Modules***. Each of them consists of a number of Program Units which represent the execution part of CIM-OSA elementary Function Models (i.e. *Functional Operations - FO's*).

However, the support for the integration of the existing ***proprietary (non-CIM-OSA conformed) CIM-Modules*** is indeed very important for the manufacturing industries in building a CIM system. From the pragmatic viewpoint, the integration of the proprietary CIM-Modules, e.g. the CAD applications for the product design, should focus more on the information integration than on the business activities integration. CIM-OSA offers the *Application Front-End (AF)* for the control of these types of proprietary applications to enable them to access the common data stored in external Data Base Management Systems. This strategy, however, complicates the AF design, because the Application Front-End not only has to achieve the capabilities of the Machine Front-End but also should include a Client-Server relationship with these proprietary applications.

An alternative solution to this problem is to add another component into the Integrating Infrastructure. This component should manage several interface adapters, such that each one is responsible for the interfacing between a proprietary CIM-Module and the Information Services. Figure 8.1 depicts a *revised CIM-OSA*

Integrating Infrastructure. It aims at the integration of both business activities and information. The external CIM-Modules are therefore divided into two types: the ***CIM-OSA conformed CIM-Modules*** and the ***proprietary CIM-Modules***. They are integrated into a CIM-OSA system by the *Front-End Services* and the ***Interface Adapter***, respectively, which allows the co-existence of both types of CIM-Modules. It is conceived that such a CIM-OSA system will be more widely applied by allowing the execution of proprietary CIM-Modules without the control of the Business Services.

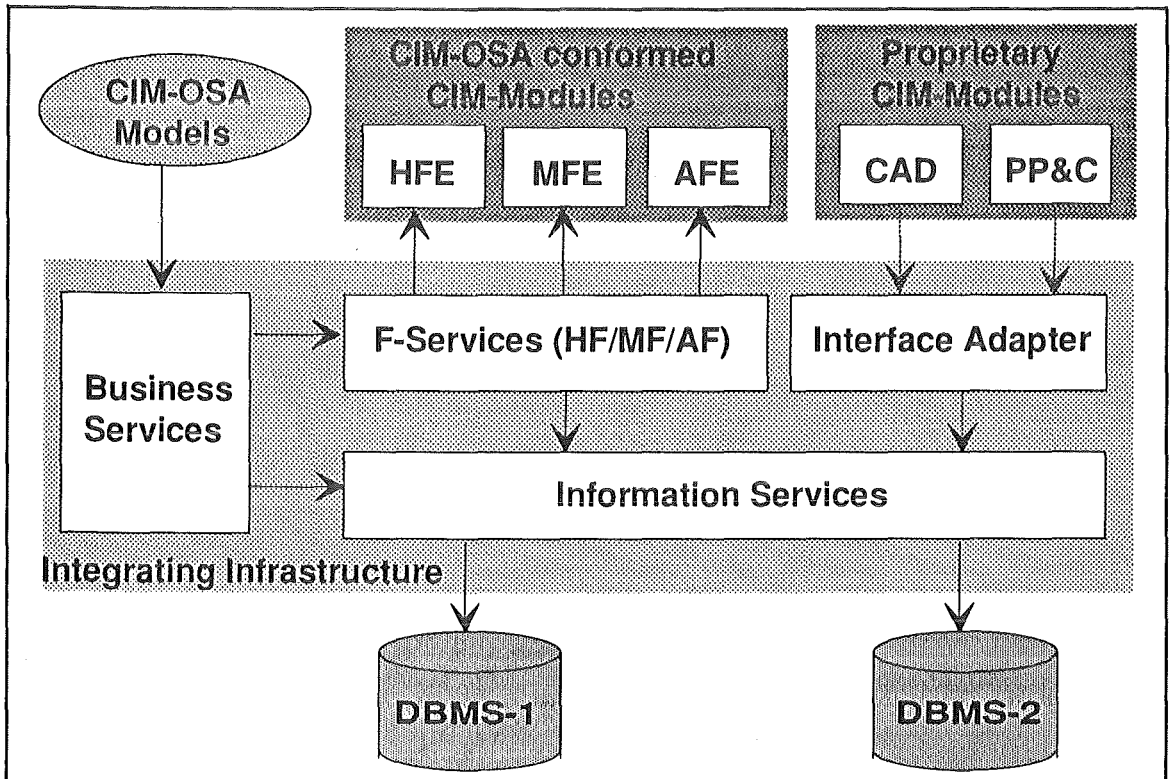


Fig. 8.1 A revised CIM-OSA Integrating Infrastructure

Keys are:
HFE: Human Functional Entity,
AFE: Application Functional Entitiy,
PP&C: Production Planning & Control,
HF: Human Front-End,
AF: Application Front-End,
MFE: Machine Functional Entity,
CAD: Computer Aided Design,
F-Services: Front-End Services,
MF: Machine Front-End,
DBMS: Data Base Mgmt. System.

Integration of the proprietary machine control systems into CIM-OSA systems

CIM-OSA provides enterprises with a *Modelling Framework* and an *Integrating Infrastructure* for the software development for the generation and operation of ***new***

(sub)systems. However, the protection of ***existing*** investments and therefore the integration of existing applications to a CIM system is an important factor when considering a new investment in a CIM-OSA system.

Generally, there are two methodologies for the integration of existing applications: the ***integration by interfacing*** and the ***integration by migration***. The integration by interfacing has been shown in the form of Interface Adapter in the previous recommendation above.

Migration of an application into a given system, means the change of the application structure in conformance with the structure of the given system in order to achieve an integrated system. Migration of a proprietary (distributed) application into an open system is a difficult task. It has to deal with the functional procedures and information objects of the proprietary application. There are several migration processes suggested: the analysis of the proprietary application software, the extraction of the distributed functions, the transformation of the distributed functions, and the testing [*Hou 91*]. The migration of a proprietary application into a CIM-OSA system needs to design a Domain Process Model which contains the procedural functions of the proprietary application.

The approach of the MF design uses the *Control Models* and the *Service Units* to control the MFO executions. Each *Service Unit* is used for a specific purpose to co-operate with remote machine controllers. Therefore, a *Service Unit*, together with its co-operating Program Unit of the machine controller, can be viewed as a basic operational unit for a specified job. In fact, these basic operational units provide a good basis not only for the implementation of Function Models, but also for the migration of the distributed functions of proprietary applications.

The ***integration by interfacing*** concentrates only on the information exchange of the proprietary parts with the CIM-OSA applications. The proprietary applications remain almost unchanged. By the short-term strategy of an enterprise, this kind of integration can achieve an integrated working system in a short time, but from the long-term aspect it loses the benefits of a CIM-OSA system.

9) Conclusions

The development of a concept for the *Machine Front-End* (*MF*) design was initiated by the study and analysis of the CIM-OSA concept, from which the two main MF objectives have been derived:

- the support of the execution of CIM-OSA models resulting from the *Modelling Framework*;
- the integration of heterogeneous manufacturing devices.

In the Client-Server Architecture of the *Integrating Infrastructure* (*IIS*), the Machine Front-End has three features:

- a twofold function, as a client and as a server;
- a provision of specific services for the other IIS components;
- an application program for the CIM-OSA users, not a service provider.

The conventional approach, consisting in building the client and server applications with a set of indication/confirmation functions, was investigated and found to be not applicable for the design of the Machine Front-End. Therefore, an advanced approach was proposed, which can achieve the two objectives and realize the three features of the Machine Front-End.

The proposed approach provides a *Control Model Libary* and a *Control Engine*. The *Control Model Libary* contains the application-specific control knowledge of the CIM-OSA models, while the *Control Engine* holds the generic control mechanism. The approach makes use of the concept of *Service Units* which may invoke the services of the international standard MMS (*Manufacturing Message Specification*) or any proprietary service. It applies the *object modelling technique* to specify the MF capabilities, and uses the principle of the complementary interaction model to define the interaction between the Machine Front-End and its clients.

Based on these concepts, a *Control Engine* was specified. Also a command language was developed for the support of the implementation of the elementary CIM-OSA Function Models, and the implemented models are contained in the *Control Model Libary*. Furthermore, a MF prototype, a testing environment and three CIM-OSA models (i.e. Domain Process Model which describes the task of a selected

working area within an enterprise) were created for the validation of this approach and the investigation of the MF behaviour in a CIM-OSA system.

The CIM-OSA models created are presently only for 'laboratory investigation', they include ,however, the subsets of the two selected industrial applications of the ESPRIT-VOICE Project. A more complex and realistic CIM-OSA model of manufacturing environment for the extensive testing of the Machine Front-End requires a complete implementation of the Integrating Infrastructure (IIS). However, due to the current state of the IIS development, much effort is still needed in the design of the specification and the implementation.

The important results of this work can be summarized as follows:

- An advanced approach for the MF design was proposed which conforms to the CIM-OSA concept and satisfies the industrial user requirements. The separation of the application-specific control knowledge of CIM-OSA models from the generic control mechanism, not only provides a great flexibility in the implementation of the CIM-OSA elementary Function Models, but also supports their portability;
- It was shown that it is possible to establish a link between the process design and the manufacturing. The implemented Test Scenario is the first CIM-OSA demonstrator which shows the CIM-OSA executable model. The scenario was registered as the McCIM system by KfK.
- Through the implementation of the Machine Front-End in the McCIM system, the CIM-OSA concept was enhanced. This work has made a considerable contribution to the evolution, acceptance and credibility of the CIM-OSA concept.

Both the proposed approach of the MF design and the created MF prototype were accepted by the ESPRIT-AMICE Consortium and recognized as being one of the important contributions to the whole CIM-OSA development.

LITERATURE

[AEG 90] AEG-GeAmatics Production Control System; AEG; '90.

[AMIC89] ESPRIT Consortium AMICE: "Open System Architecture for CIM", ESPRIT Project 688 AMICE Volume 1, '89, Springer-Verlag.

[AMIC90] Technical annex of ESPRIT Project no. 5288. AMICE II/M Consortium, Sept. '90, Brussels/Belgium.

[Ansa89] "ANSA: An Engineer's Introduction to the Architecture" and "The ANSA Reference Manual"; Architecture Projects Management Limited, Poseidon House, Castle Park, Cambridge, CB3 ORD, U.K. '89.

[Appe85] H.-J. Appelrath: "Von Datenbanken zu Expertensystemen"; Informatik-Fachberichte 102, Springer-Verlag, '85.

[BaKH89] Baumgartner, H., Knischewski, K., Wieding, H.: "CIM-Basisbetrachtungen", Siemens AG, '89.

[Beec90] Beeckman,D.:"CIM-OSA - An Illustrative example of how to apply the Modelling Framework";CIMCOM Conf.,Gaithersburg USA,22-24 May '90.

[Borl92] Bordland Co.: "Borland C++: Programmierhandbuch", Feb. '92.

[CIMO90] CIM-OSA: "CIM-OSA Documents:
C5-1000 Communication Services Complex;
C5-2000 Infomation Services Complex;
C5-3000 Front-End Services Complex;
C4-1000 & C4-2000 Modelling Framework and IIS;
B0,B1,B2,B3 Modelling"; '90-'91.

[CODE90] Technical annex of ESPRIT Project no. 5499. CODE Consortium, Sept. '90, Brussels/Belgium.

[COMP89] COMPUTROL: "COMPUTROL MAP 3.0 Software Manual for MS-DOS/PC-DOPS"; July '89.

[CONC90a] CONCORD: "OSI Programmer's Package: Programmer's Reference Manual"; July'90.

[CONC90b] CONCORD: "MAP-Board Controller User's Manual"; April '90.

[ChDa89] Chan,B.; Daly,D.: "The MAP Solution in the General Motors Oshawa, Ontario Car Body Assembly Plant"; SPIE Vol. 1179 Fiber Networking and Telecommunications, Sept. '89.

[Craw93] Crawford, R.:"Integrating 3D modelling and process planning by features: a case study"; Int. J. Computer Integrated Manufacturing, Vol.6, NOS. 1 & 2, p113-118, '93;

[CYCL92] Cycle Software Inc., 60 Kinnaird St., Cambridge, MA 02139, USA; User's Guide of CYCLE LiveData: An Open Networking Tool for Real-Time Data, Version 1.06.

[Das 92] Das, S.K.: "A Scheme for classifying integration types in CIM"; Int. Journal Computer Integrated Manufacturing, Vol. 5, No. 1, p10-17, '92.

[DEC 91] Digital Equipemt Coporation: "Neue Dimensionen in der flexiblen Fertigung mit CIM", "BASEstar", "Digital's Integrated Manufacturing Management Solution", '90-'91.

[Didi92] Didic,M.: "Rapid Prototyping for MAP/MMS based CIM-OSA Environment". The 3rd Int. Workshop on Rapid System Prototyping. North Carlonia, USA, June 23-25, '92.

[DiNe91] Didic,M.; Neuscheler,F.: "Analysis and Requirements for Tool Supplements and Additions for the ESPRIT Project VOICE"; KfK/IAI; Primärbericht; Dec. '91.

[DNBM93] Didic,M.; Neuscheler,F.; Bogdanowicz,L.; Klittich,M.: "McCIM: Execution of CIMOSA Models"; Proc. of the 9th CIM-Europe Annual Conf., 12-12 May '93, Amsterdam, The Netherlands.

[Exce90] Excel -- User's Guide; Version 3.0; Microsoft Corporation, '90.

[Espr92] "Esprit: Computer-Integrated Manufacturing and Engineering"; "Esprit: : Information Processing System and Software" and "Esprit: Advanced Business and Home Systems -Peripherals"; The Synposes; Commission of the European Comminities; Oct. '92.

[EWIC83] Techniques for Verification and Validation of Safety Related Software, EWICS TC7 WP 400, May '83.

[EWIC84] Guidelines for Verification and Validation of Safety Related Software, EWICS TC7 WP 333, October '84.

[Flat88] Flatau,U.: "CIM Architecture Framework", Digital Equipment Corporation, Digital Competence Center Manufacturing Industries Europe, Munich, Germany, '88.

[GHLS91] Grandjacques,B.; Hou,W.-N.; Leuschner,R; Shi,H.; Segarra,G.; Bruel,P: "Front-End Services Feasibility Report", VOICE Project, June, '91.

[GiBH91] Gielingh,W.F.; de Bruijn,W.J., Halbert,J.R. : "Implementation Levels for Semantic Integration of Open System CIM Modules"; Proc. of the 7th CIM-Europe Annual Conf., 29-31 May '91, Turin, Italy.

[Guit92] Guittot,Ph.: "Entwicklung einer Anwendung zur Demonstration der CIM-OSA-Konzepte"; KfK internal report; '92.

[GoSp90] Gora,W.; Speyerer,R : "ASN.1 (Abstract Syntax Notation One)"; DATACOM Verlag; '90.

[Gyme93] Gymer,P.: "VOICE - ELVAL Simulation in InTouch"; KfK/IAI, internal report; Feb. '93.

[Hars92] Hars,A; et al.: "Reference Models for Data Engineering in CIM"; Proc. of the 8th CIM-Europe Annual Conf., 27-29 May '92, Birmingham, UK.

[Holl91] Holler,H. E; et al.: "Untersuchung der Anwendbarkeit terrestrischer Standards zur Produktionsautomatisierung auf Weltraumanwendungen"; Primärbericht, KfK/IAI, Germany, Oct. '91.

[Hou 87] Hou,W.-N.: "Entwicklung eines Prototypen für ein Expertensystem zur Diagnose und Reparatur von Steuerungssystemen in der Zementindustrie"; Holderbank Management & Beratung AG/Switzerland; '87.

[Hou 91] Hou,W.-N.: "A Migration Platform for the Introduction of Open Communication into a CIM-Environment (McCIM)"; KfK/IAI internal report, '91.

[HoKG92] Hou,W.-N.; Klittich,M.; van Gerwen,R.: "The Knowledge Base-oriented Machine FRont-End Services for the Integrating Infrastructure of CIM-Open System Architecture (CIM-OSA)"; Proc. of the 8th CIM-Europe Annual Conf.; 27-29 May '92; Birmingham, UK.

[Hou 93a] Hou, W.-N.: "An Approach to the Development of the Machine Front-End Services in a CIM-OSA Environment"; Proc. of the 3rd Int. Conf. on Flexible Automation and Integrated Manufacturing; 28-30, June, Limerick, Ireland.

[Hou 93b] Hou, W.-N.: "Specification of the Machine Front-End of the CIM-OSA Integrating Infrastructure"; KfK/IAI internal report, June '93.

[Hug 91] Hug, H.: "Die Kommunikations-Infrastruktur in der Fertigung"; CIM-Management; 1/'91.

[IBM 91] International Business Machines: "IBM CIM Architecture", "CIM Communication and Data Facility", "IBM Computer Integrated Manufacturing", '90-'91.

[InTo92] InTouch - the Application Generator for the Man-Machine-Interface; User's Guide; Version 3.21, WonderWare Software Development Corporation, '91.

[ISO 87] ISO 8824: "Specification of Abstract Syntax Notation One (ASN.1)"; ISO 8825: "Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)"; May '87.

[ISO 90] ISO 9506: "Manufacturing Message Specification (MMS): Part 1: Service Definition and Part 2: Protocols Definition"; '90.

[Jack86] Jackson, P.: "Introduction to Expert Systems"; Addison-Wesly, '86.

[JoVe90a] Jorysz, H.R.; Vernadat,F.: "CIM-OSA Part I: Total Enterprise Modelling and Function View"; International Journal of CIM, Vol 3 # 3/4 '90.

[JoVe90b] Jorysz, H.R.; Vernadat,F.: "CIM-OSA Part II: Information View"; International Journal of CIM, Vol 3 # 3/4 '90.

[KaCE92] Kathawala,Y.;Chawla,S.; Elmuti,D.: "Some Strategic Aspects of Computer Integrated Manufacturing"; Integrated Manufacturing System, Vol.3 No. 1, '92, pp.27-34.

[Kais89] Kaiser, F.-J.: "MAP-Pilotinstallation: Integrations- und Interoperabilitäts-tests für eine zeitkritische Umgebung", KfK-Primärbericht, Juli '89.

[Kern89] Kerndlmaier, M; et al.: "Technische Expertensysteme für Prozeßführung und Diagnose"; Oldenbourg Verlag; '89.

[Klit90] Klittich,M.: "CIM-OSA Integrating Infrastructure-The operational Basis for Integrated Manufacturing Systems"; Int. Journal of CIM,Vol 3 # 3/4 '90.

[Klit91a] Klittich,M.: " From CIM to CIM-OSA -a step ahead in system integration"; Proc. of the 7th CIM-Europe Annual Conf.; 29-31 May '91; Turin, Italy.

[Klit91b] Klittich,M.: "Front-End Service Candidates", internal paper of AMICE Project, Nov. '91.

[Kosa90] Kosanke,K: "CIM-OSA: its role in Manufacturing Control"; Proceeding of the 11th IFAC World Congress,TALLIN,ESTONIA,USSR;Aug. '90.

[Kosa91] Kosake,K.: "Open Systems Architecture for CIM: Standards for Manu-facturing"; Proc. of the Int. Conf. on CIM, 2-4 Oct. '91; Sigapore.

[Lawo90] Lawo,M.; et al.: "Automatisierung- und Steuerungskonzept für ein hochflexibles Handhabungssystem zum Gußputzen"; KfK/IAI internal report, '90.

[MAP 89] MAP: "Manufacturing Automation Protocol Specification"; Version 3.0; Vol.1-4, European MAP Users Group(EMUG); '89.

[Meie91] Meier, A: "Advantages of Using Features to Integrate Product and Process Modelling - Results of IMPPACT (ESPRIT 2165)"; Proc. of the 7th CIM-Europe Annual Conf.; 29-31 May '91; Turin, Italy.

[Micr90] Microsoft Windows Software Development Kit: "Guide to Programming"; "Reference - Volume 1 and 2"; Microsoft Corporation, '90.

[Mits90] User Guide of Robot RV-M1; Mitsubishi, Japan; '90.

[Neus90] Neuscheler, F.: "ESA/ESTEC-Report: Survey of Automation and Robotic Relevant Standards"; KfK/IAI-Report; '90.

[PACE91] "PACE Tool Reference Manual"; Grossenbacher Elektonik AG; St. Gallen, Switzerland, '91.

[Pans90a] Panse,R.: "CIM-OSA - A Vendor Independent CIM Architecture (Integrating Infrastructure)"; Maple Conference; Ottawa Canada, May 15, '90.

[Pans90b] Panse,R: "CIM-OSA - A Vendor Independent CIM Architecture (Modelling Approach)"; Proc. of CIMCOM Conf.; Gaithersburg USA, 22-24 May '90.

[PeSi84] Peterson,J.; Silberschatz,A.: "Operating System Concepts"; Addison-Wesley Publ.; '84.

[Pime90] Pimentel, J.-R.: "An Object Oriented Environment for Intelligent Automation Systems"; Int. Conf. on Industrial Electronics, Asilomar, CA., Nov. '90.

[PROC90] PROCOS A/S: "EasyMap User's Manual"; Nov. '90.

[Quir85] Quirk,W.J.: "Verification and Validation of Real-Time Software"; Springer-Verlag; '85.

[Quer91] Querenet, B.: "The CIM-OSA integrating infrastructure" Computing & Control Engineering Journal; May '91.

[Rafi92] Rafiee,N.:"Process Tracking für Renault"; KfK/IAI;internal report;Dec.'92.

[ReDi86] Rembold,U.; Dillmann,R.: "Computer-Aided Design and Manufacturing"; Springer-Verlag; '86.

[Redm88] Redmill, R.J.: "Dependability of Critical Computer Systems 1"; Elesevier Science Publisher Ltd.; U.K.; '88.

[Remb90a] Rembold,U.: "Standardization Efforts in Computer Integrated Manufacturing"; Proceeding of the 14th All India Machine Tool Design & Research Conference, Bombay, India; Dec. 19-21 '90.

[Remb90b] Rembold,U.; et al.: "CAM-Handbuch"; Springer-Verlag; '90.

[RoWL83] Hayes-Roth,F.; Waterman,D.A.; Lenat,D.B.: "Building Expert Systems"; Addison-Wesly; '83.

[Sche89] Scheer, A.-W.: "Enterprise-wide Data Modelling"; Information Systems in Industry; Berlin, '89.

[Sche90] Scheer, A.-W.: "CIM: Der computergesteuerte Industriebetrieb"; Springer-Verlag; '90.

[Sche91] Scheer, A.-W.: "Architektur integrieter Informationssysteme"; Springer-Verlag; '91.

[SISC90] SISCO: "MMS-EASE Refernce Manual"; Revision 9; System Integration Specialists Company; '90.

[Stra89] Strasser,T.D.: "MAP/TOP - Overcoming Barriers to Integration"; Journal of Electrical and Electronics Engineering, Vol. 9, No. 4, '89.

[SUN 90] Sun microsystems: "Network Programming Guide"; March, '90.

[Vern90] Vernadat, F.: "Modelling and Analysis of Enterprise Information Systems with CIM-OSA"; Proc. of the CIM-Enrope 6th Annual Conf.; Lisbon Portugal; 15-17 May '90.

[Visu91] Visual Basic -- User's Guide & Language Reference; Microsoft Corporation, '91.

[Vlie90] Vlietstra,J.: "The Architectural Framework and Models in CIM-OSA"; APMS (Advanced Production Management Systems) Conference, Helsinki, 21 August '90.

[VOIC90] Technical Annex of ESPRIT VOICE Project 5510, Oct. '90.

[VOIC93] Final report of ESPRIT VOICE Project 5510, Feb., '93.

[Weat88] Weatherall, A.: "Computer Integrated Manufacturing: From fundamentals to implementation"; Butterworths Publishing Co.; '88.

[West90] Weston, R.H.; et al: "Highly Extendable CIM Systems Based on an Integration Platform"; Proc. of CIMCON Conf.; Gaithersburg, USA; '90

[West91] Weston, R.H.: "CIM Enterprises of the 21st Century"; Proc. of ICCIM Conf.; Singapore, 2-4 Oct. '91;

[West91] Weston, R.H.; et al: "'Soft' Integration and its Importance in Design to Manufacturing"; Journal of Design and Manufacturing; 1; p47-56; '91.

[WiHo89] Winston,P.-H.; Horn,B.-K.: "LISP 3rd Edition"; Addition-Wesley; '89.

[WZL 90] "Education & Training Course on CIM-Open System Architecture"; Laboratorium für Werkzeugmaschinen und Betriebslehre (WZL); TH Aachen; Nov. '90.

[XuPa90] Xu,J.; Parnas,D.L.: "Scheduling Processes with Release Time, Deadlines, Precedence, and Exclusion Relations"; IEEE Transactions on Software Engineering, Vol.16, No.3, March, '90;