7-1-2000

IDL-XML based information sharing model for enterprise integration

Uanny Brens Garcia

IDL-XML Based Information Sharing Model for Enterprise Integration

Uanny M. Brens Garcia

B.S. Pontificia Universidad Catolica Madre y Maestra
(Santiago, Dominican Republic 1998)

Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the Department of Industrial and Manufacturing Engineering in the Kate Gleason College of Engineering of the Rochester Institute of Technology

July 2000

KATE GLEASON COLLEGE OF ENGINEERING

ROCHESTER INSTITUTE OF TECHNOLOGY

ROCHESTER, NEW YORK

CERTIFICATE OF APPROVAL

MASTER OF SCIENCE DEGREE THESIS

The M. S. Degree Thesis of Uanny M. Brens Garcia
has been examined and approved by the thesis committee
as satisfactory for the thesis requirement for the
Master of Science degree.

Sudhakar Paidy, Ph.D. *Advisor*

Wayne Walter, Ph.D.

Permission granted

IDL-XML Based Information Sharing Model for Enterprise Integration

I, Uanny M. Brens Garcia, hereby grant permission to the Wallace Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Date: 07/25/00 Signature of Author: _____

Dedication

To the Brens family and the García Family. They have taught me to never give up, to always have faith in myself

Acknowledgment

Throughout the time I have been working in this research study, many people have assisted me. It is impossible to acknowledge them all, nevertheless, I would like to specifically thank the following individuals:

Dr. Sudhakar Paidy, for all the time he dedicated to my research work and for the knowledge he has shared with me in the last two years. This thesis would not have been possible without your help.

Dr. Wayne Walter, for his support and comments on my work. I hope you enjoyed being a member of my thesis committee.

The faculty and staff of the Industrial and Manufacturing Engineering Department, not only for their academic support and feedback on my work, but for all the advise given throughout my career at RIT.

Dr. James Miller, Dr. Jasper Shealy and the officials at PUCMM for setting up the program that allowed me to pursue thesis Master of Science Degree.

To Anne, Kevin, Kaine, Sri, Judith, Seau, Tara, Jenny, Joy and Tom for helping me from the beginning of the thesis, proofreading, giving me feedback and support. But mostly, for your friendship, all of you have a special place in my heart.

Outline

Section **Page**

Tables and Figures

APPENDIX

1. Abstract

CIM is a mechanized approach to problem solving in an enterprise. Its basis is intercommunication between information systems, in order to provide faster and more effective decision making process. These results help minimize human error, improve overall productivity and guarantee customer satisfaction.

Most enterprises or corporations started implementing integration by adopting automated solutions in a particular process, department, or area, in isolation from the rest of the physical or intelligent process resulting in the incapability for systems and equipment to share information with each other and with other computer systems. The goal in a manufacturing environment is to have a set of systems that will interact seamlessly with each other within a heterogeneous object framework overcoming the many barriers (language, platforms, and even physical location) that do not grant information sharing.

This study identifies the data needs of several information systems of a corporation and proposes a conceptual model to improve the information sharing process and thus Computer Integrated Manufacturing.

The architecture proposed in this work provides a methodology for data storage, data retrieval, and data processing in order to provide integration at the enterprise level. There are four layers of interaction in the proposed IXA architecture. The name IXA (IDL – XML Architecture for Enterprise Integration) is derived from the standards and technologies used to define the layers and corresponding functions of each layer. The first layer addresses the systems and applications responsible for data manipulation. The second layer provides the interface definitions to facilitate the interaction between the applications on the first layer. The third layer is where data would be structured using XML to be stored and the fourth layer is a central repository and its database management system.

2. Introduction:

Since their introduction, computers (and computer-assisted applications) have been developed to better serve humans. They have been widely used throughout manufacturing environments. In 1953, General Electric was one of the first organizations to use computers in financial applications. One year later, numerical control machines were introduced; and then in 1955, the first automatically programmed tool started what we now know as Computer-Aided Manufacturing - CAM (Shrensker, 1991).

After the 1950's, computers have become faster, smaller and more affordable. This resulted in numerous computer applications for different manufacturing tasks. Computer Aided Design was made public in the early 1960's and other CAM software like Materials Requirement Planning (MRP) had their first appearance in the early 1970's (Robinson, 2000). MRP is a software developed to help with scheduling tasks, but it has turned out to have relatively poor performance (Robinson, 2000). The next generation of Computer Integrated Manufacturing (CIM) was Manufacturing Resource Planning, or MRPII. It was with these two applications (MRP, and MRPII) that the term Computer Integrated Manufacturing was coined. Dr. Joseph Harrington first used the term in 1974, but it was not until 1981 that it became widely used, probably after MRP and MRPII became popular (Shrensker, 1991).

Since 1981, computers have evolved from having a couple of hundreds of kilobytes of memory to several gigabytes of memory (among other resource enhancements), making them very powerful and enabling many computer-aided systems to be developed. For example, computer-aided engineering, computer-aided systems for manufacturing process planning, shop scheduling, inventory control, decision support, human resources applications, and diverse financial applications were making common appearances in 1980s within manufacturing environments (Shrensker, 1991). Today, applications are run not only on dedicated systems, but are run across multiple computers and across networks. They are as complex as mainframes or as small as sensors and logic controllers; and the manufacturing floor includes varied computerized devices such as robots and robotic cells, computer numerical controlled machines, automated decision support systems, etc.

Computer Integrated Manufacturing comprises not only the product making activities such as engineering and manufacturing but all the disciplines within an enterprise including marketing, sales, distribution, finance, human resources, and general office administrations. According to Shrensker: "A fully integrated CIM system involves the design, development, or application of each of the systems in such a manner that the output of one system serves as the input of another" (Shrensker, 1991). In other words, Computer Integrated Manufacturing is an enterprise-wide effort, where the data would "flow" from one place to another and improve the business process.

The Society of Manufacturing Engineers defines CIM as the integration of the total manufacturing enterprise through the use of integrated systems and data communications coupled with new managerial philosophies that improve organizational and personnel efficiency (Rehg, 1994). It is a mechanized approach to problem solving within an enterprise. In the early

1980's the general concept of Computer Aided Manufacturing lead to what is known as Islands of Automation. Computer Integrated Manufacturing's aim was to integrate these independent systems (Vernadat, 1996). Bartlett explains that CIM is concerned with providing computer assistance and control, along with high-level integrated automation at all levels of the manufacturing industries, by linking islands of automation into a distributed processing system (Bartlett, 1995).

Many companies in the United States and most companies in developing countries still handle the output of automated systems with manual techniques. There remains a gap between when the information is needed and when it can be delivered (this is based on the time it takes to collect the data manually). The solution should be a responsive, adaptable, and reliable computer systems that supports the CIM concept throughout the enterprise (Attran, 1995).

The basis of Computer Integrated Manufacturing is intercommunication between the information systems processors to facilitate a faster and more efficient decision making process. Processors at all decision making and supervisory levels of an enterprise need the ability to gather the data and track the status of operations necessary to assess manufacturing efficiency (Snyder, 1997). Specialized software, plus the integration of data systems, provides the typical firm with a much more robust system than has been possible in the past (Oleson, 1999).

The problem faced with CIM is nicely described by Bubsy, when he explains that the principal computer applications in a manufacturing company are large and substantially isolated systems bought from different suppliers and originally specified to be freestanding of other systems. They run on different types of machines, use different operating systems, and they are generally built on different database management systems (Bubsy, 1992).

But Bubsy is not the only author concerned with the poor integration of systems in an enterprise. Associations such as the Society of Manufacturing Engineers (SME), the Manufacturing Executions Systems Association (MESA International), and companies like Boeing devote their time to research the current technology and to develop new technology that would improve the integration in the enterprise. Many researches have also expressed the need for applications and information systems to have seamless communication in order to achieve full enterprise integration (Allegri, 1989; Hardwick, 1996; McClellan, 1997; Adelsberger, 1995; Scharpf, 1999; Singh, 1996; and Svenson, 1993).

The importance of information and data sharing through the manufacturing facility lies in the fact that accurate and consistent information is crucial to make business decisions and process improvement (Bhatt, 2000). Since the last quarter of the Twentieth Century, automation has forced manufacturing companies to develop more and more sophisticated computer communication strategies in order to guarantee their maximum benefit to the company (Bartlett, 1995). The availability of better data, advances in analytical techniques, and ever improving communications enhance a company's ability to make more complex and effective decisions, and thus maintains a company's competitive edge (Metz, 1998).

Information is not only needed in an accurate manner, it is also needed in a timely fashion. Real-time industrial information is crucial for process control. Therefore, information needs to be

real-time, accurate, reliable, flexible, and modifiable to interface directly with the departments that have the responsibility of controlling the processes, and in predicting the results of their actions and decisions (Rathmill, 1998). Information Systems (IS) and Information Technology solutions, through communication networks and database systems, enable organizations to create and sustain process improvement (Bhatt, 2000). The tools exist to develop technologies and to acquire integration throughout the enterprise. Today, most information systems and systems integration vendors work toward the development of an enterprise-wide system that contains a central database compiled from the operations of multiple machines and multiple processes (Snyder, 1997). Companies like Adasoft Group say that a factory must be considered as a whole, where the systems for administration, management, maintenance control, follow up and command cannot be independent (Adasoft Group, 2000).

In order for manufacturing applications to communicate efficiently, an information structure is needed. Although the nature of the information is diverse in manufacturing environments, this information is frequently common to other applications (Busby, 1992; Hardwick, 1996). There is the need of structured information sharing methods and models that would allow data to be used by the different applications and systems. Bubsy makes the point that if no mechanisms are introduced to specifically regulate the information sharing process, there is a risk of inconsistency, duplication and omission of data (Bubsy, 1992).

The Computer Integrated Manufacturing Architecture (CIM Architecture) explained in following sections gives a schema of how information flows in a manufacturing environment. In the same way, the Manufacturing Execution Systems Association's Context Model (MESA Context Model) defines the six information systems relevant to an enterprise and relates these systems according to the data shared. These two architectures share many of the basic concepts of information flow and we present a diagram that relates these two architectures.

Furthermore, the relationship between the data of an enterprise and the six information systems (Enterprise Resource Planning, Supply Chain Management, Sales and Service Management, Product and Process Engineering, Manufacturing Execution Systems and Controls) is graphically shown in tables. These tables have the objective of reinforcing the concept that information is common to different departments and systems of an enterprise. Thus, a good information sharing architecture should be implemented to improve the data sharing process.

The objective of this thesis work is to propose an architecture that provides a methodology of data storage, data retrieval, and data processing in order to provide integration at the enterprise level. The model developed its called **IDL-XML** Architecture for Enterprise Integration (**IXA** for Enterprise Integration). There are four layers of interaction in the proposed IXA architecture. The name **IXA** Enterprise Integration is derived from the standards and technologies used to define the layers and corresponding functions of each layer.

The two technologies used to develop the **IXA** are the Common Object Request Broker Architecture (CORBA), and the eXtensible Markup Language (XML). CORBA is a set of standards that establishes the interfaces and parameters for applications to interact with one another. One of CORBA's components is the Interface Definition Language (IDL) which is the description of an object's characteristics to facilitate the interactions of different components.

The XML is a markup language that defines how data would be structure; it gives the control of how the data would be displayed and manipulated to the application using the data.

Therefore, CORBA and XML are the basis of the IDL-XML Architecture for Enterprise Integration proposed in this work. The purpose of the **IXA** model is to facilitate the information storage, retrieval, and processing of information across the enterprise. Since data are collected and utilized from systems of different platforms, languages, applications, allocations, etc., it makes the process of information access, utilization, and sharing difficult. By using a central repository, or common database storage, there would be a place for all hardware and/or software in the corporation to store and retrieve information when needed. Interaction between the central repository and the information system would be controlled and manipulated by layers of software that would act as an intermediate access manager, and would restrict the information sharing and access process. The concept is to develop interface layers (based on CORBA and XML) between the systems and the central repository.

The **IXA** model would provide the means for communication among information systems via CORBA. It structures the data using XML hierarchy to allow a better navigation through digitally represented documents and information, and provides an API interface (using CORBA's IDL) to process information between client and server.

3. Computer Integrated Manufacturing

3.1 Description and Background

Computer Integrated Manufacturing is a modern manufacturing paradigm. It aims at integration of man and machine activities by facilitating communication, cooperation and coordination of the various technical, administrative, and support functions of a manufacturing company. It aims to achieve this by means of information and communication technologies, from design to production planning, from control to manufacturing and shipping (Vernadat, 1996). The importance of data communications is further emphasized by the Society of Manufacturing Engineers; SME states that data communication improves organizational and personnel efficiency in a corporation.

Usually, solutions developed to improve an area involve a single automation tool. This concept is known as "Islands of Automation" and it was the most prevailing model of computer integrated manufacturing in the 1980's. As a result of this, automated solutions are implemented in a particular process, department or area, in isolation from the rest of the physical or intelligent process. However, the key is to approach enterprise's problems with an integrated set of tools that will consider and fix all the problems at once and improve over all production (Vernadat, 1996). This is what is known as Computer Integrated Manufacturing (CIM).

Computer Integrated Manufacturing (CIM) became the solution for an integrated set of problems recognized by companies. The Society of Manufacturing Engineers (SME) defines CIM as the interaction of the total manufacturing enterprise through the use of integrated systems and data communications coupled with new managerial philosophies that improve organizational and personnel efficiency (Rehg, 1994). In other words, Computer Integrated Manufacturing's goal is to integrate man and machine activities by facilitating communication, cooperation and coordination of the various technical, administrative and support functions of a manufacturing company, by means of information and communication technologies between design, planning, control and shipping (Vernadad. 1996). Thus, Computer Integrated Manufacturing is a mechanized approach to problem solving in an enterprise. Its basis is intercommunication between information systems to provide faster and more effective decision making process, and as a result minimizes human error, improves overall productivity and guarantees customer satisfaction.

3.2 CIM Architecture

Information flow in an enterprise is crucial to determine the design and implementation of an integration system. By referring to the National Institute of Standard and Technology (NIST) model for manufacturing plants, we notice the hierarchy of the different activities within a company. If we follow this model to structure the information flow, so that each layer communicates only with the layer directly above or directly below, the sharing and flow of the information will occur in a more simplified and rapid way. (OMG, 1998)

The NIST developed the Shop Floor Production Model (SFPM) as a generic architecture for real-time production control. It is based on a tree-shaped hierarchy with command/feedback control structure, where the control processes are isolated by function and communicate via standard interfaces (Jones et al, 1986). This approach ensures that the size, functionality and complexity of individual control modules are limited. Each of the modules in this architecture has to be designed in a way that humans can comprehend and interact with them. Also, hierarchical approach reduces the complexity of the control problems because the control resides at the next higher level and the decisions are made at the lowest possible level (Jones, 1986; Rathmill, 1988). Every control module decomposes its current input command from its supervisor into procedures to be executed at that level, subcommands to be issued to one or more lower modules, and status feedback sent back to the higher level (Jones, 1986).

The SFPM hierarchy has been developed based on control modules, each of which is determined by its planning horizon. The planing horizon is the period of time over which the module is responsible for planing and updating local goals. The main principle behind SFMP architecture is that processing becomes more and more time critical at the lower levels of the hierarchy; thus, the planing horizon becomes smaller at the lower levels and the processing becomes more time critical. (Rathmill, 1988).

The different levels of the NIST CIMS architecture are:

Level 5 – Facility Control System (Also called: Plant level, Enterprise Level, Factory Level):

This is the highest level of control and its implemented in the front office. The planning horizon can be anywhere from several months to years. Its responsibility is for overall planing and execution; it manages the production of finished products by controlling job shops. Its decisions are strategic in nature, and of great importance for the corporation. This level gives support to the CIM system through the integration of orders, sales data, manufacturing, shipment, and invoicing.

Factory level is broken down in subsystems that fall into three major functional areas:

Manufacturing engineering: Here are the functions typically carried out with human involvement via user data interfaces (i.e. CAD/CAM).

Information management: Provides user data interfaces to support necessary administrative or business management functions (i.e. cost estimation, job cost accounting, inventory accounting, etc).

Production management: Its function is to track major projects, generate long-range schedules, identify production resources, etc.

Level 4 – Shop Control System (Also called Factory level)

The shop control system manages the production of parts and part's component by controlling work centers. It coordinates the production and support activities that are carried out by the cell controllers at the next lower level. This level is responsible for coordinating the production and support jobs on the shop floor, and it is also responsible for allocation of resources to those jobs. However, the main function of this level is to schedule production and provide management information by monitoring and supervising the lower levels of control.

The decisions of the shop control system are both strategic and operational, and it has a planing horizon that could be anywhere from several weeks to several months.

Two major components of this level are:

Task manager: Schedules job orders, equipment maintenance, and shop support services; it also tracks equipment utilization, schedules preventive maintenance, and material transfer equipment in the factory.

Resource Management: It allocates workstations, buffer storage areas, trays, tooling, and materials to cell level control systems for particular production jobs. It also monitors and updates levels for all raw stock, work in process, cutting tools, and replacement parts inventories necessary to run the factory.

Level 3 – Cell Control System (Also called: Area Level, and Cell Level):

Cell control system's responsibility is the sequencing of batch jobs of similar parts though workstations and supervising other support services, for example material handling, or calibration. This level is in charge of tasks like goods storage and retrieval, and transportation and manufacturing of parts. It directs and coordinates the shop floor production process, but also, coordinates production flow among various stations and integrates individual stations into an automated system. Modules within the work cell level perform task decomposition, analyze resource requirements, prepare requisitions, report job progress and system status to the shop control system, make dynamic batch routing decisions, and schedule operations. Also it decodes the order received from the higher level into more elementary manufacturing jobs so that machine cells can process the information.

The planning horizon can be anywhere from several hours to several weeks, and the decisions it takes are operational in nature.

Level 2 - Workstation Control System (Also called: Work-center Level, or Station Level):

Workstation level coordinates and directs the activities of small, integrated physical groupings of shop floor equipment, and comprises a group of machines.

This level is in charge of converting the input from lower levels to output commands, since it exchanges data with the scheduling, production planning and material handling activities and at the same time with the controllers, actuators, and sensors from the lower levels.

At the Workstation Control System the planning horizon can be anywhere between several minutes to several hours, and the nature of the decisions is operational.

Level 1 – Equipment Control System (Also called Machine level, or Equipment level):

Here is where basic interface with plant floor equipment takes place. The main tasks of the Equipment Control System are to translate the commands from the workstation controller into a sequence of simple tasks that can be understood by the vendor supplied controller and to monitor the execution of those tasks via the various sensors attached to the hardware.

The equipment control system acts as the interface between the workstation control system above and the vendor supplied controller on the hardware under its control. Therefore, the information gathered comes from the execution of the job at the shop floor. The planning horizon may be anywhere between several milliseconds to several minutes.

Level 0 – Floor level:

This level is comprised of all the devices that perform direct action on the manufacturing process, i.e. sensors, and actuators. Their task is to collect information related to the execution of the job at the shop floor, and to carry out the commands in order to keep the manufacturing process running. They have the smallest planning horizon and fastest processing and decision making time (Allegri, 1989; Jones, 1986; Rathmill, 1988; Vernadat, 1996; OMG 1998).

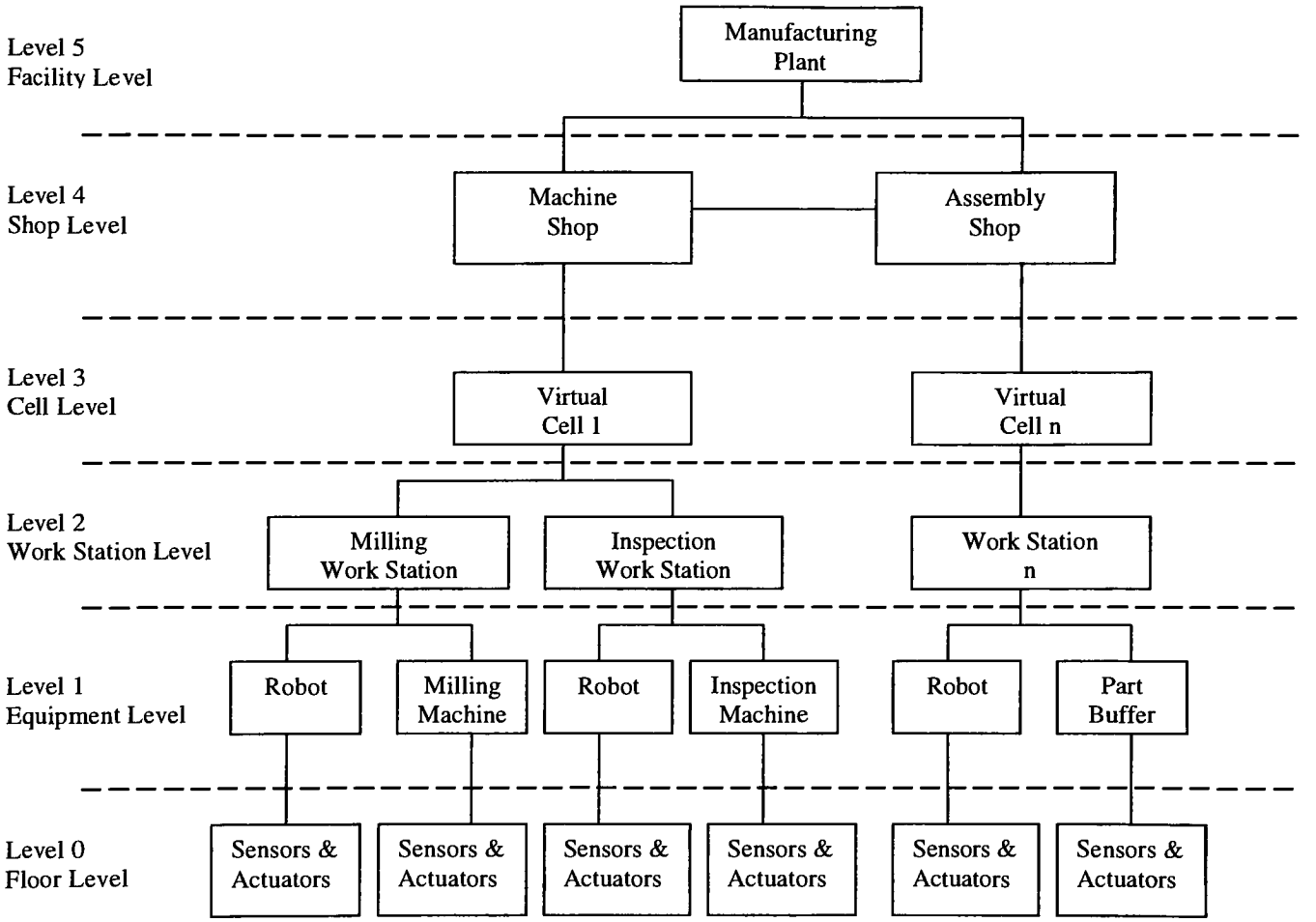


Figure 3.1: The NBS Model for manufacturing plants (Vernadat, 1996)

3.3 Manufacturing Execution Systems Association

3.3.1 MESA Information Systems Architecture:

The Manufacturing Execution Systems Association International (MESA International) has developed an Enterprise Information System Architecture or Context Model. This model depicts the relation between the major manufacturing software systems categories. Each system (Enterprise Resource Planning, Manufacturing Execution System, Supply Chain Management, Sales and Service Management, Product and Process Engineering, and Controls) is the grouping of different applications and functions of an enterprise. Thus, an enterprise needs some functionality from each of these systems in order to achieve success (MESA 1997). The scope and utilization of the systems will depend on the type of enterprise and manufacturing process it has (if it is a service provider, a discrete manufacturing, continuous, batch, or assembly process, make to order, etc).

MESA gives the following definitions for each system:

Enterprise Resource Planning (ERP): Consists of those systems that provide financial, order management, production, and material planning. ERP is a manufacturing information system whose focus goes beyond inventory control and distributed management activities. Its purpose is to bring together areas like engineering, finance, human resources, project management, etc. Moreover, it can be consider as a medium to achieve the connection via computer systems, between the different subsets of manufacturing systems. ERP is more concerned about automation and computer-aided processes of the integration, and hence is the information system that automates core activities of an enterprise.

Sales and Service Management (SSM): Comprises software to support the sales activities, as well as after sales service. It takes care of sales automation, product configurations, services quoting, product return, and so forth.

Supply Chain Management (SCM): Includes functions as forecasting, distribution and logistics, transportation management, e-commerce, and advanced planning systems. Supply chain management is the logical customer-focused progression of physical distribution and logistics management, it incorporates the material flow functions of receiving raw material or sub assemblies, manufacturing, distribution and delivering (Metz, 1999). It is a network of facilities and distribution options that performs the functions of procurement of materials, transformation of these materials into intermediate and finished products, and the distribution of these finished products to customers (Ganeshan, 2000).

Product and Process Engineering (P&PE): Includes computer aided designs and manufacturing, process modeling, and product data management. P&PE keeps track of the life cycle for both the products and the process. It is the tool that captures the learning that goes on with the product and process. The system collects data from the process and turns it into information about the process. It then analyzes the information and determines what course of action to take (Oleson, 1998).

Manufacturing Execution Systems (MES): The Information System that provides information to effectively execute operations in order to meet business goals. Using current and accurate data, MES guides, initiates, responds to and reports on plant activities as the occur. The resulting rapid response to change gives as an output effective plant operations and processes. MES provides mission-critical information about production activities across the enterprise and supply chain via bi-directional communications

Controls: Systems and computerized process controls designed to monitor and adjust the way in which products are being manufactured. Control systems are necessary to collect and process information about certain tasks and attributes in order to maintain the manufacturing process running in accordance to what is expected and to make the appropriate decisions when required (Vaughn, 1967).

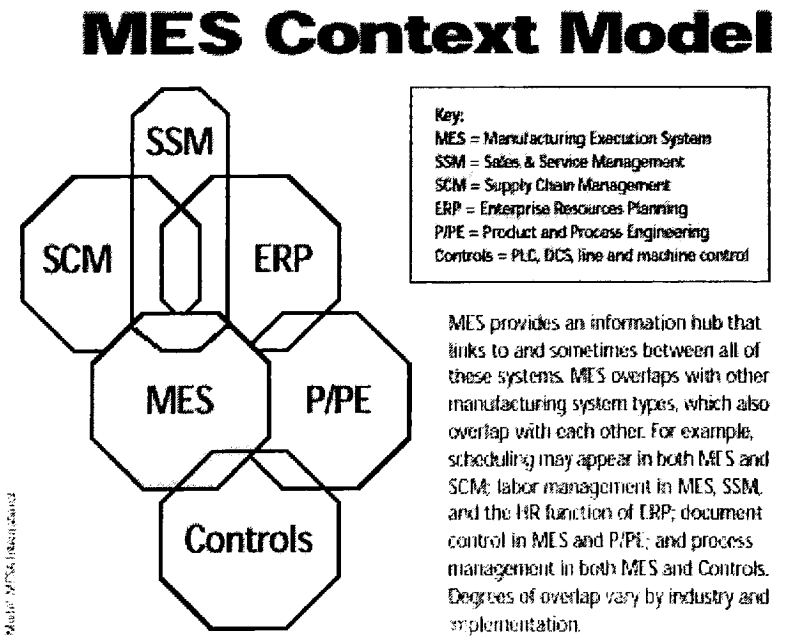


Figure 3.2: MES Context Model (MESA, 1997)

After a close look at both the MESA Information Context Model and the NIST CIM Architecture model, a similarity in functionality can be noted. Each model describes the relation between systems and applications within a manufacturing environment; both depict similar type of implementation in their respective model.

The tasks or functions of an enterprise are classified according to similarity of information shared, type of resources used, and the significance of the output with other functions in the enterprise. The CIM model positions the activities involved in job shop production hierarchically with respect to management activity and area of responsibility (Vernadat, 1996). The MESA model relates the activities on an enterprise according to the information shared.

Similarities between the two models confirm the flow of information within an enterprise, where all aspects of manufacturing and front office functions have to be considered, but where communication in all directions among all systems is not possible. This communication incompatibility is due to the different nature of the interacting functions, and is the cause of the emerging manufacturing integration systems with object oriented functions and distributed computing.

Based on the similarity of information flow, we could map the MESA system functions to the CIMS architecture. This would give us an idea of the area of application each function on the MESA system addresses.

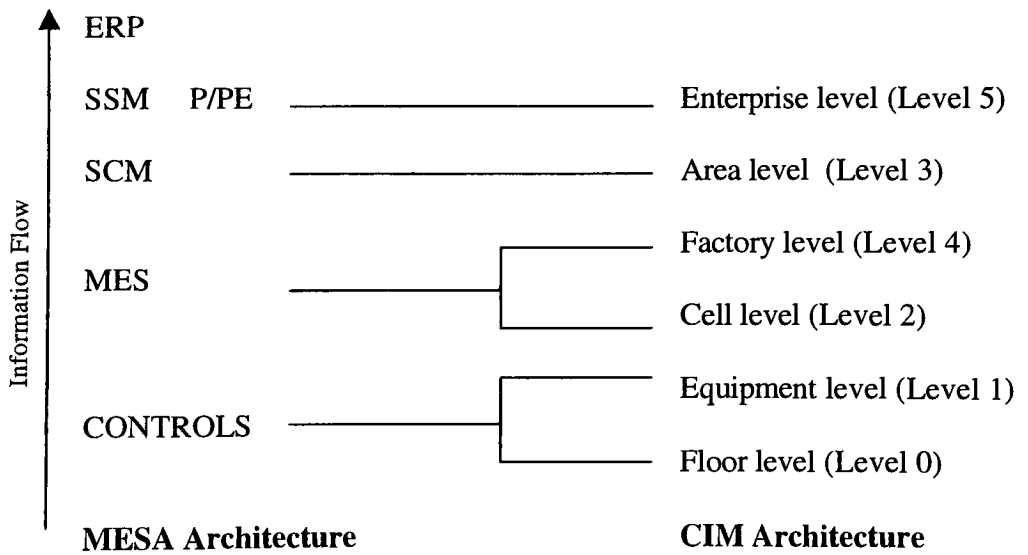


Figure 3.3: Relationship between MESA architecture and CIMS architecture

The components of the Floor and Equipment level (sensors, actuators and controllers) monitor, regulate and manipulate, in a direct manner, the manufacturing operation. The Control System would manage these devices, and would share the necessary information from the process with the MES.

The Cell level needs the scheduling and production planning information in order to handle the proper instructions to the lower levels. The Shop Control System is where that production planning is processed. Moreover, MES is the "virtual supervisor". It looks over the controllers and actuators to make sure the process is running accordingly; it prepares the scheduling and keeps production records.

While SCM consists of forecasting, logistics and advanced planning; the Area Level makes the sequence of jobs. Coordinates production low among different entities of the enterprise, the retrieval of storage material, and the transportation of parts.

The Enterprise level is concerned with the overall process, without looking at the small details. Since SSM and P&PE need to look at the process, product, market, sales and suppliers, it also needs an overall view of the enterprise.

ERP, as stated in the next pages, comprises all departments and functions. It takes into account information from each of the departments and each of the information systems.

3.3.2. Enterprise Resource Planning:

According to Chang (Chang, 1998), Manufacturing System is an organization that comprises several interrelated manufacturing subsets. Its objective is to interface with outside production functions in order to optimize the total productivity performance of the system (production time, costs, and machine utilization). The activities of these subsets include design, planning, manufacturing, and control. These subsets are also connected with production functions outside the system, such as: accounting, marketing, financing, and personnel.

Enterprise Resource Planning (ERP) is a manufacturing system whose focus goes beyond inventory control and distributed management activities. ERP can be considered as a medium to achieve the connection, via computer systems, between the different subsets of manufacturing. Production decisions affect and are affected by areas like engineering, accounting, human resource, marketing, and sales, and in order to make better decisions, corporations have to take into account the impact those areas have. By making available the necessary information, ERP systems assure an enhanced decisions making process (Hicks, 1995).

It is worth noting that ERP is the result of the evolution of other manufacturing systems, more particularly, Materials Requirement Planning (MRP) and Manufacturing Resource Planning (MRP II). MRP systems translated the master schedule built for the end items into time-phases net requirements for the sub-assemblies, components and raw materials planning, and procurement. MRP II evolved as an extension of MRP to shop floor and distribution management activities. ERP is the "natural" evolution of these systems, when areas as

engineering, finance, human resource, project management and others were added. The firsts to coin this term, ERP, were the Gartner Group of Stamford, Connecticut (Hicks, 1995).

Although some authors describe ERP as a collection of software programs, and others as a set of applications, the common concept is that ERP system integrates various enterprise functions, automating front office departments' functions, and making information available throughout the corporation. "ERP attempts to integrate all departments and functions across a company onto a single computer system that can serve all those different departments' particular needs" (Koch, 1999).

Since integration began as a set of Islands of Integration (where different software programs serve the purposes of the different departments and operations within a corporation) each works independently from the others, and thus ERP seeks to unify these systems. ERP aims to create a computer environment where all systems of an enterprise or corporation share information between each other; all applications within the system have access to a common database, giving as result a process more effective and less prone to error. ERP software defines a specific application for a specific operation (i.e. payroll, customer order, scheduling, and product request...), "everyone in the company sees the same computer screen and has access to the single database that holds the customer's new order" (Koch, 1999). ERP leans on client/server distributed architecture, RDBMS (relational database management systems) and object oriented programming technology. The information is accessible by who ever need it. "ERP objective is to take information from every corporate function, it is a tool that assists employees and managers to plan, monitor, and control the entire business" (Cambashi, 1999).

The main goal of ERP systems is to allow the corporation to gain control of their dispersed information sources, incompatible computer systems and multilingual work force. Also, ERP provides:

- Information access: features and functions that deliver fast, accurate information to users to enable them to make sound decisions.
- Productivity: architecture and tools that increases worker productivity by automating processes and tasks.
- Freedom of choice (open architecture): support for leading systems platforms, relational databases, and operating systems.
- Flexibility: tools that are easy to use, and applications that are easy to customize and easy to upgrade.

With ERP companies will see integration of enterprise wide information and the elimination of redundant data. ERP software is designed to model and automate many of the basic processes of a company, from finance to the shop floor, with the goal of integrating information across the company and eliminating complex, expensive links between computer systems that were never meant to interact with each others.

3.3.3 Manufacturing Execution Systems

MESA International gives the following definition of MES, which is widely accepted through out the different MES vendors and enterprises:

"Manufacturing Execution Systems deliver information enabling the optimization of production activities from order launch to finished goods. Using current and accurate data, MES guides, initiates, responds to and reports on plant activities as the occur. The resulting rapid response to changing conditions, coupled with a focus on reducing non-value-added activities, drives effective plant operations and processes. MES improves the return on operational assets as well as on-time delivery, inventory turns, gross margin, and cash flow performance. MES provides mission-critical information about production activities across the enterprise and supply chain via bi-directional communications" (Davis, 1999; mesa.org).

Implementation of ERP systems involves more than the top-level activities of a corporation. But in order for a ERP system to be of benefit for a corporation, the manufacturing part of the corporation, (the plant floor, production, and quality data) must be integrated to the system. Here is where MES comes to play.

MES is the link between controls and ERP. According to Adasoft Group, "A factory must be considered as a whole. The systems for administration, management, maintenance control, follow up and command cannot be independent" (www.adasoft.com). MES works as the bridge between front office accounting and the factory supervisory control systems and product, and its the part of the enterprise's information systems that resides on the plant floor.

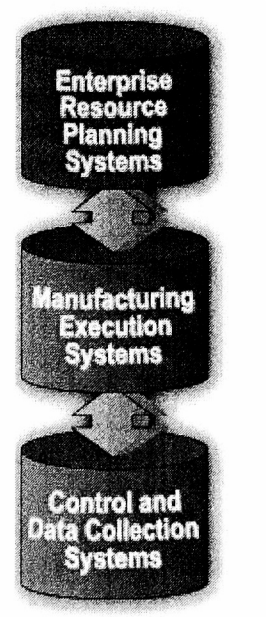


Figure 3.4: Flow of information (Raytheon Automated Systems; http://www.esys.com/rec/autosys/sub_erp.htm)

MES provides basic interface between planning and execution systems (MESA, 1997). This interface enables a two-way communication between administrative offices and factory floor information, aiding the office systems in job costing, payroll, lot control, inventory, and other

factors on work being performed. But also, interaction between MES and the Control System is very important: MES is in charge of the overall flow of the product or process. The specific performance of a piece of equipment, or an operator, is not as important to the MES as to a controller. That is why, MES' central repository collects data from various locations within the factory, rather than from a particular area. The MES downloads instructions about the process operators and machines required to carryout the manufacturing operations (See figure 3.5).

Akin ERP, MES is a product of the evolution of MRP and MRP II. MES is viewed by some as an enhanced version of the functions performed by MRP II (MESA, 1997). The first manufacturing applications occurred as a by-product of the computerization of accounting, more specific, inventory accounting. MRP II introduced a closed-loop manufacturing system, emphasized in material planning. But MES improved the coordination of cross-functional activities, the development of a realistic schedule, and most important, the development of applications with the capability to monitor the floor activities and to continuously analyze activities in order to be responsive to events as they occur.

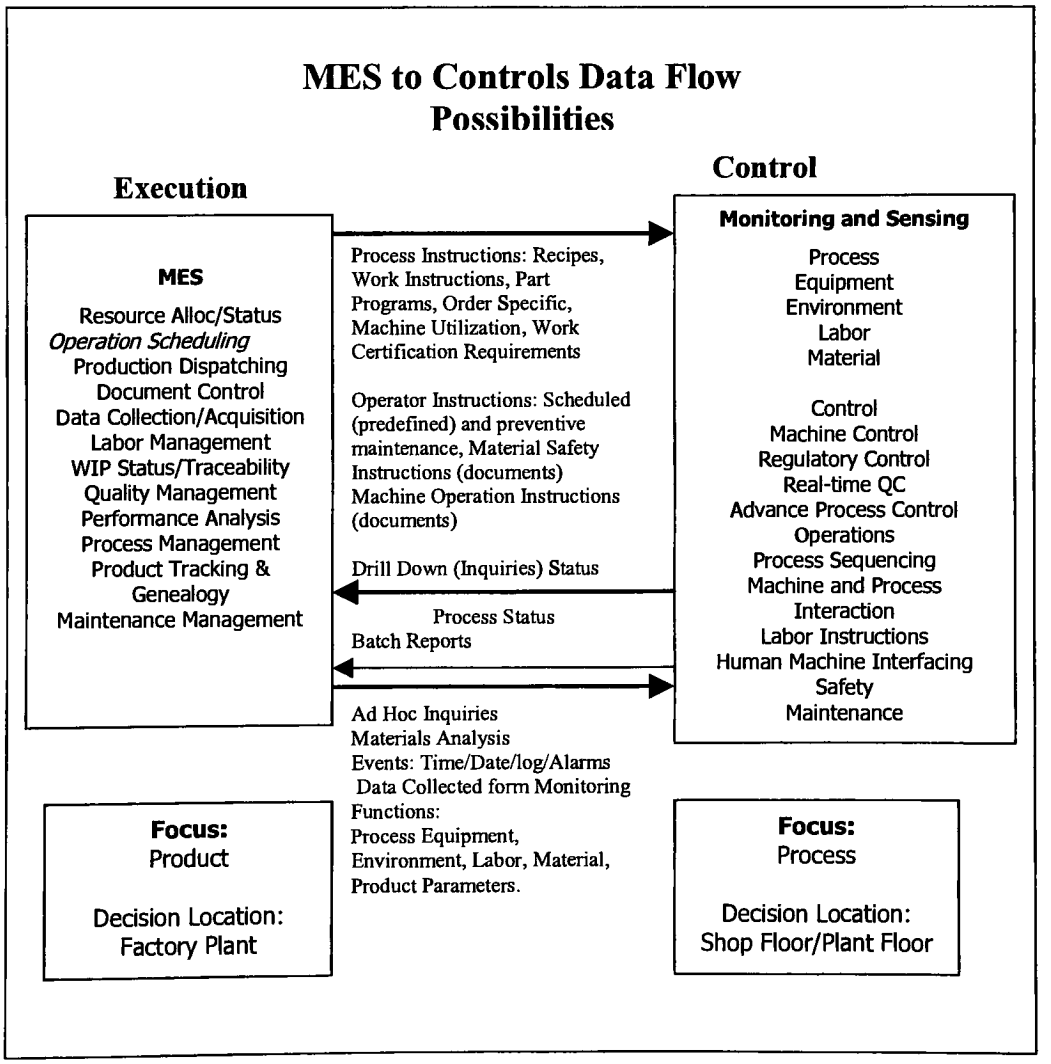


Figure 3.5: MES to Controls Data Flow Possibilities (MESA, 1997)

The primary goal of any manufacturing execution system is to accelerate the flow of work throughout the plant floor. MES provides an electronic network for performance improvement because it can monitor production in real-time, giving accurate and recent information to solve situations and avoid production delays or quality problems (MESA, 1997). Otherwise, production control personnel and supervisors may take many hours a day to research the current status of all the jobs on the floor in order to solve a problem or to develop a work schedule (Davis, 1999). MES delivers information that enables the optimization of production activities from order launch to finished goods. Thus MES provides critical information about the production across the enterprise (MESA, 1997).

MES provides a virtual production keeper, supervisor, quality control and material requesters. It dumps the necessary data and information into a common database within a computerized system that gives back the status of the production and the plant floor. This system can be seen as a virtual counterpart of the physical process, by which one can monitor the products in a more efficient and accurate way. Is a real-time display of the factory performance that provides the manufacturer with a look at the shop floor it can really use to manage production (MESA, 2000). The system's core functionality is based around tracking work-in-process (WIP) through detailed product routing, labor reporting, resource and rework management, production measurement and data collection (Fulcher, 1999).

One of MESA reports describes Manufacturing Execution Systems as the capture of production knowledge that oversees and records results of activities in a production facility. It gives a plant-wide view of the status and operation of processes, materials, human resources, machines and tooling, and describes the MES system as a set of different activities that work together to create the system (MESA, 1997). According to this organization, these activities are:

Resource allocation and status: Manages resources that must be available in order for work to start at the operation. It provides history of resources and status in real time. It guides what people, machines, tools and materials should do, and tracks what they are currently doing or have just done.

Operations/Detailed Scheduling: Provides sequencing based on priorities, attributes, and characteristics that when scheduled in sequence properly minimizes set-up. These sequencing and timing activities for optimized plant performance are based on finite capacities of the resources.

Dispatching Production Units: Manages flow of production units in the form of jobs, orders, batches, lots, and work orders. Dispatched information is presented in the sequence in which the work needs to be done and changes in real time as the events occur on the factory floor. It also has the ability to control the amount of work in process at any point with buffer management. In other words, it gives the command to send materials or orders to certain parts of the plant to begin a process or step.

Document Control: Controls records or forms that must be maintained with the production unit (work instructions, recipes, drawings, standard operation procedures, part programs, batch records, engineering change notices, shift-to-shift communications). Stores historical data, and gathers certification statements of work and conditions.

Data Collection/Acquisition: This function provides a link to obtain operational and production data. This data may be collected from the factory floor either manually or automatically from equipment in an up-to-the-minute time frame. It monitors, gathers, and organizes data about the processes, materials, and operations from people, machines, or controls.

Labor Management: Provides status of personnel, including time and attendance reporting, certification tracking, and the ability to track indirect activities as basis for activity based costing. Tracks and directs the use of operations personnel during a shift based on qualifications, work patterns, and business needs.

Quality management: Provides real time analysis of measurements collected from manufacturing to assure proper product quality control and to identify problems requiring attention. Records, tracks, and analyses product and process characteristics against engineering ideals.

Process Management: Monitors production and automatically corrects or provides decision support to operators for correcting and improving in-process activities. It provides interfaces between intelligent equipment and MES through Data Collection/Acquisition. Directs the flow of work in the plant based on planned and actual production activities.

Maintenance Management: Tracks and directs the activities to maintain the equipment and tools to insure their availability for manufacturing and insure scheduling for periodic or preventive maintenance as well as the response (alarms) to immediate problems. It maintains a history of past events or problems to aid in diagnosing problems.

Product Tracking and Genealogy: Provides the visibility to where work is at all times and its disposition, this record allows traceability of components and usage of each end product. It monitors the progress in units, batches, or lots of outputs to create a full history of the product.

Performance Analysis: Provides up-to-the-minute reporting of actual manufacturing operations results along with comparisons to past history, goals and metrics set by the corporation, customers, or regulatory bodies.

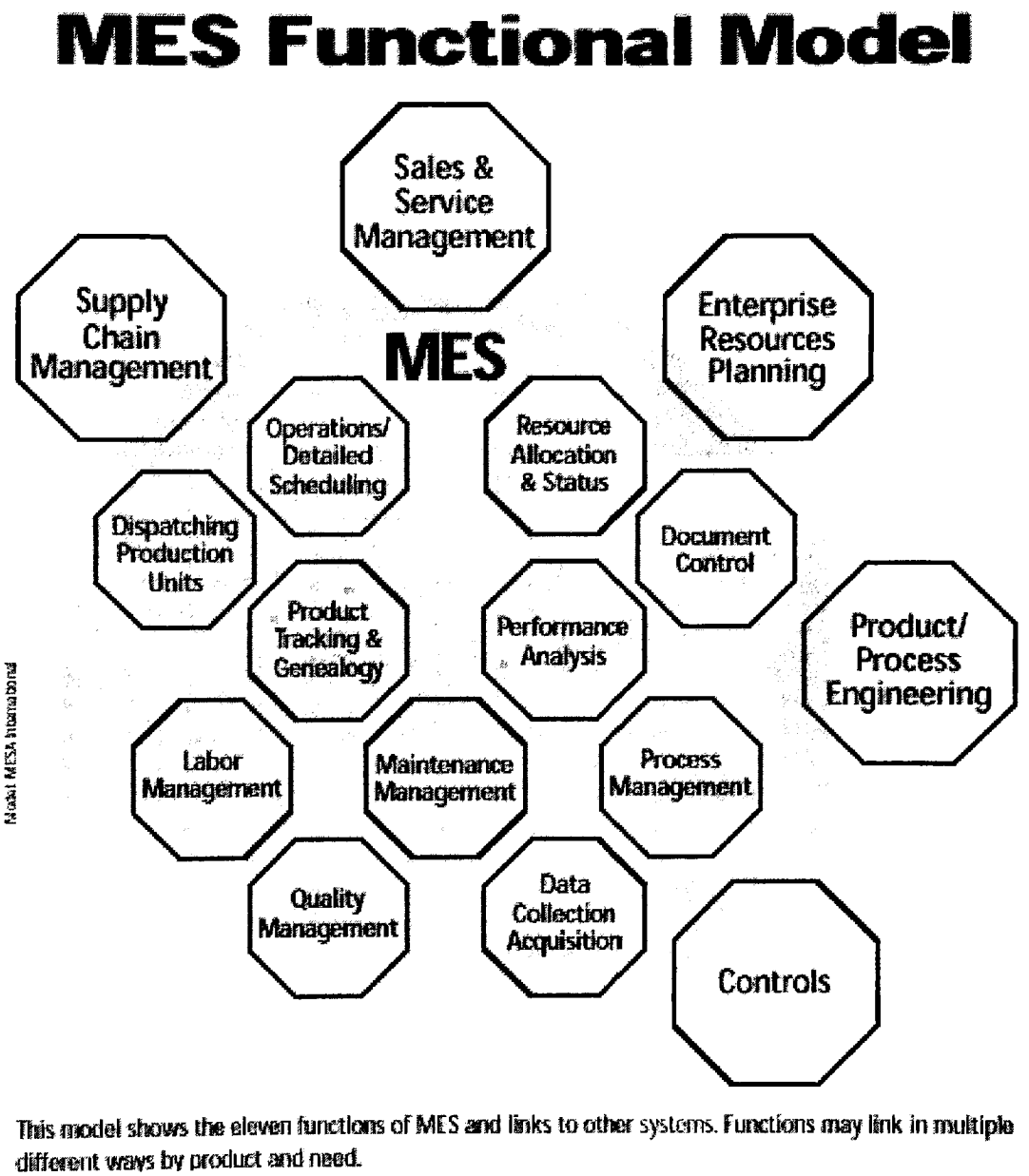


Figure 3.6: MES functions (MESA, 1997)

3.3.4 Control Systems

A rough definition of Control Systems is a system in which one or more outputs are forced to change in a desired manner as time progresses. Its the entity responsible for measuring, monitoring, and manipulating production, people, products and processes (MESA, 1995). Control Systems corresponds to the factory floor/process system. The function of it is the use of all the equipment on the factory floor (hardware, software, and people) in a manner consistent with the goal of producing a product or process that falls within the parameters set forth by the corporation (MESA, 1997). It collects data from the available resources in order to maintain characteristics of the output constant with a standard.

The function of control people (and control machines) is to monitor and control their own operations. To assure that the outputs are in compliance with the requirements set by the coordinator: "controls work in real-time, there are always operations occurring to refine, or correct the process to maintain desired tolerances" (MESA, 1995). Some of the resources used by the control systems are: Programmable Logic Controllers (PLC), Robots, DCS, sensing devices, Computer Numerical Controls (CNC), user operator interfaces, display devices, special software, and intelligent input devices.

Hence, Control Systems focus on sequencing and manipulating the process to assure that tolerances are kept within defined limits, that material flow is maintained and that all of the people, equipment, and resources involved within the process are fully utilized. Control is only concerned about the inputs and outputs (or status points) of the process, in order to deliver a product within specifications and a process without complications.

There are different types of control:
Direct control, any control that is effected by the structure or preset setting of the control device, without adjustment related to performance.

Feedback control system, its one that varies its control on output in accordance with a comparison of the output with a standard. This type of control can be: *closed loop control*, if the system is completely mechanical and the feedback is part of the mechanical systems; or *open loop control*, if there is a gap in the mechanical connection that is filled by human interaction. Furthermore, it can be classified into continuous or intermittent control, depending on the continuity of the testing and correction of the system (Blair, 1971).

In conclusion, control systems are necessary to collect and process information about certain tasks and attributes in order to maintain the manufacturing process running according to what is expected and to make the appropriate decisions when required (Vaughn, 1967).

3.4 CORBA, STEP, DCOM and Information Sharing

One has to realize that the common element to the Information Systems of an Enterprise (MES, SCM, SSM, P&PE, Controls, and ERP) is the data. It does not matter which system is going to interact with each other, or from which level on the architecture information comes from, sooner or later, that stored data is going to be used. The information will be saved in a database, and it is this part of the enterprise information system that has to be accessible to everyone.

Furthermore, the information has to be encoded in such a way that facilitates the use of it by any program and any client. Here is where the issue begins: the information system has to be robust enough to permit any type of data to be process (either received, send, or manipulated) on any type of program, from any where in the planet, and in any possible platform. So far, implementations of different data types in some programs or by some languages are limited; communication between hardware (or software) that are not compatible is difficult; and even physical distance limits the communication between systems or networks.

Standards like CORBA (Common Object Request Broker), DCOM (Distributed Component Object Model), and STEP (STandard for the Exchange of Product Model Data), are grouping the requirements and architecture necessary to create applications capable of interacting between programs despite the different barriers. They could be seen as interpreters, but they are more, because these middleware (as they are often called) act as requesters between a client and a service.

STEP, CORBA and DCOM differ in subtle things, but the core of each is based on the same principle: to create a medium that would allow interaction between different applications. From these three standards, CORBA has been developed (and thus adopted) by more than 200 commercial software developers, organized in what is called the Object Management Group.

The developer of DCOM is the Microsoft Corporation. A Distributed Component Object Model (DCOM) is a model for creating software such that every application is constructed from a set of components, each of which performs a set of predefined tasks. These components may be distributed across a network, or be on the same machine. Ideally, a DCOM allows components to be invoked transparently, i.e., as though they resided on the same machine as the invoking component. Components can also interact dynamically at run-time, without requiring a recompilation to interact with newly created components (www.gsia.cmu.edu/bb26/70-456/projects/java/dcom.htm).

The Standard for the Exchange of Product Model Data (STEP) is an ISO standards project to develop mechanisms for the representation and exchange of a computerized model of a product in a neutral form. The goal is to enable a product representation to be exchanged without any loss of completeness or integrity (Bradford Smith, NIST; web.cad.gatech.edu/~peak/step).

However, even if there is not a universal standard for information exchange, there has to be a relationship between the developed standards in order to achieve the goal: to process data in a system independently of software, hardware or language. The integration of STEP and the CORBA standards promises to satisfy urgent needs in the industrial sector, as well as to enhance

the two standards communities. While STEP gains a new implementation method in the form of distributed objects, CORBA gains facility that serves the broad domain of product information (Hardwick, 1996).

3.4.1 CORBA Definition

The Object Management Group -OMG- with representations of more than two hundred software companies (since 1989) has been developing standards and technology for object oriented distributed applications under the name Common Object Request Broker Architecture -CORBA- (Seetharaman, 1998; Orfali, 1997; Otte, 1996).

CORBA is a set of standards or protocols for communication among heterogeneous systems (Murphy, 1998, Seetharaman, 1998; Orfali, 1997). CORBA gathers the specifications from the Object Oriented technology. The goal is for every software or computer application to be designed and developed under the CORBA specifications, and thus, be capable of sharing information with one another. The architecture described using CORBA allows interaction of components independent of the designers, the platform they are running on, the language they are written in, and where they are executed (Seetharaman, 1998; Siegel, 1998).

The IDL language and the ORB interfaces are the main parts of CORBA, and are described in the following segments:

Object Request Broker (ORB): The Object Request Broker (ORB) is software that acts as an intermediate between the client and the server, it manages the client requests and hides the location and implementation details of the server from the client (Seetharaman, 1998). The ORB is that object bus where requests are made, and responses received, from objects located locally or remotely (Orfali, 1997).

A CORBA object models a real-world object and consists of data and methods that may be invoked on it (Seetharaman, 1998). These objects are packaged as binary components that remote clients can access via method invocations Because of this object infrastructure, it is easier for components to be more autonomous, self-managing and collaborative (Orfali, 1997).

Interface Definition Language (IDL): IDL is necessary to code information of an object into a set of interface definitions. This interface definition specifies the operations the object is prepared to perform, the input and output parameter each requires, and any exceptions that may be generated along the way. With IDL developers can define interfaces and data structures that can later be retrieved using a high level language (Otte, 1996; Siegel, 1998).

As Siegel explains, the OMG IDL compiler uses mapping specifications to generate a set of method invocations from the IDL operations. Developers refer to the OMG IDL file (this contains the definitions of interfaces the client or server implementation will support) and use the language mappings to generate the corresponding set of method declaration. After compilation the skeleton is ready to make the right calls to invoke operations on the object implementation (Siegel, 1998).

OMG IDL is used to define interfaces and data structures, but not to write algorithms. IDL is use to generate source code for the desired programming languages. At this moment IDL has language-mapping specifications for C, C++, Java, Smalltalk, and Ada. But the greatest advantage of using IDL is that it frees CORBA applications from being restricted to any particular programming language (Mowbray, 1997; Orfali, 1997; Otte, 1996; Siegel, 1998).

IDL supports location transparency, this means that a remote distributed object can not be distinguish in its invocation from a local component. Also, IDL keeps the system unaware of changes in the implementation of the software components and the physical allocation of components until run-time. This is also a feature carried out by ORBs (Mowbray, 1997).

Even though IDL is not the only way to distribute a system into useful components, it is relative simple and provides support for both industries and standard organizations (Mowbray, 1997).

3.4.1.2 Manufacturing Execution Systems and CORBA:

As stated before, communication between different software and hardware tools is a common problem not only in manufacturing, but also in many other departments of an enterprise. In the manufacturing area in particular, the multifaceted nature of design and process information makes communication particularly difficult (Hardwick, 1996). An information infrastructure is needed to allow manufacturing applications to interchange information in an efficient manner and to allow engineers to use familiar applications and tools whenever possible. However, the technology used to communicate data must be cost-effective, flexible, and portable (Hardwick, 1996), resulting in a set of standards which are the base for the new technologies.

According to Hardwick, the information infrastructure of an enterprise should have three qualities: performance, concurrency, and comprehension. **Performance** because the data needed for an operation should be delivered in a single communication, avoiding delays that additional communications may bring. **Concurrency**, data delivered should only contain the necessary information, this way others can access the rest of the information and work concurrently with the database. And **Comprehension** between the systems to assure the receiver can process the data sent (Hardwick, 1996). Systems can process data only when they understand the internal organization of the data. The complexity of manufacturing data is a key issue in manufacturing applications (Morris, 1994).

Distributed Computing and Object Oriented Programming are two important concepts in computer integration. Distributed Computing is sharing two or more pieces of software and their information which could be running on the same computer or on different computers connected to the same network (Otte, 1996). Object Oriented Programming defines objects that can be used, or reused, throughout many applications. For a system to implement distributed computing technology, the different computers have to communicate with each other, and the different software must also interchange information. Integrated information flow requires a carefully designed linkage of sets of databases and computer networks that are managed to make data flow freely and transparently between systems components (Singh, 1996).

MESA stated that object-oriented technologies are emerging as a manufacturing software direction, with the new architecture that promises easier integration, faster change, more ability to configure systems, and effective infrastructure for triggering mission critical events (MESA, 1997). Also, a distributed object framework allows data and functional logic to be carried closed to where it is used. One could conclude from these ideas that MES can be enriched from object oriented programming and distributed computing, and that its future goes toward these two trends. In fact, CORBA architecture models could be used to improve interaction between layers of the information systems of an enterprise. CORBA's Object Request Brokers allow any vendor's object written in these standards to communicate and inter-operate. This technology could allow MES to inter-operate with the other five categories of manufacturing software relatively seamlessly (MESA, 1997).

The fact that CORBA promises interaction between systems independently of language, platform and location comes to aid in the weakness of communications between the Controls System and MES. The variety of controllers that interact in a manufacturing floor, and the diverse languages used to deliver data, makes the processing of manufacturing information a complicated procedure. CORBA and its ORBs, provide a more versatile and faster way to deliver data to systems.

With this in mind, MESA has developed a model of MES architecture, which considers the Object Request Broker for the interchange of information and services between systems (MESA, 1997). In this model, the ORB is the one responsible for the location of the server within a specific request. It will manage information among different systems, creating a "transparent" communication through out the system.

3.4.2 eXtensive Markup Language (XML)

The development of XML started in 1996 by the World Wide Web Consortium (W3C), and it has been a standard since 1998. It evolved from the Standard Generalized Markup Language (SGML) which has been a standard since 1986 and HyperText Markup Language (HTML) a standard since 1990.

A definition of XML is that of a "meta language", because is a language for defining other languages (Claben, 2000). XML is a set of guidelines for designing text formats for structured data, in a way that produces files that are easy to generate and read by a computer application. These text files are unambiguous, and avoid common pitfalls such as lack of extensibility, lack of support for internationalization/localization and platform dependency. XML allow designers to customize tags and attributes, and thus enabling the definition, transmission, validation, and interpretation of data between applications and organizations (webopaedia.com, 2000).

XML's purpose is to describe a data tree. It does not define how data should be presented, only how it is structured. Data can be extracted from an XML tree using DOM, X pointers, eXtensible Style Sheet Language, and eXtensible Query Language (Minnick, 1999). W3C have described specifications for writing the text file (XML), and also further guidelines that provide specifications for other tasks. These include:

Xlink (describes ways to add hyperlinks to XML files)
Xpointers
Xfragments
CSS (style sheet language)
XSL (advance language for style sheets)
XSLT (transformation language)
DOM (set of function calls for manipulating XML files from a programming language)
XML namespaces (to associate an URL to single tags and attributes)

And several more modules that still are under development (www.w3c.com).

Even though XML files are text files, the tags and attributes it uses are meant for an application to determine the appropriate operations of such text files. XML describes a class of data objects called XML documents and partially describes the behavior of computer programs that process them. XML is an application profile, or a restricted form of SGML, but unlike SGML, XML is used for many different kinds of data not just for technical documents.

XML is useful for a browser and also for application-to-application data exchange, this could facilitate supply chain and other business communications (Claben, 2000). Also there are several application that can not be accomplished with either SGML or HTML, and may be accomplished with XML
(http://metalab.unc.edu/pub/sun-info/standards/xml/why/xmlapps.html):

1. Applications that require the Web client to mediate between two or more heterogeneous databases.
2. Applications that attempt to distribute a significant proportion of the processing load from the Web server to the Web client.
3. Applications that require the Web client to present different views of the same data to different users.
4. Applications in which intelligent Web agents attempt to tailor information discovery to the needs of individual users.

Because of the increasing need of data interchange and also the extensive use of the World Wide Web in business and in every day life, there is more and more attention given to the development of a more capable language. A language that will allow programmers to take a more active role in the development of their text files for data access. XML seems to be a tool that provides this approach, and because it is system independent, it also makes data more accessible, and easier to distribute.

3.4.2.1 XML and Enterprise Communications

The significance of XML in integration extends beyond the Web or email, database records, and programming APIs. An XML parser imposes the same API on any XML data source, eliminating much of the need for custom programs to extract and integrate information from each source. So, integrating enterprise information from accounting, purchasing, manufacturing, shipping, and other functions can be accomplished by first converting each source to XML and then processing the parsed data stream. Put in another way, each application need to know only two source formats (its own and XML) rather that having to produce the native format of every other application. XML is a universal data format that allows computers to store and transfer data that can be understood by any other computer system. Since XML maintains the content and structure, but separates the business rules form the data, each part of the information sharing process can apply its own rules (Goldfarb, 2000, Glushko, 1999).

XML by itself doesn't enable "plug-and-play commerce". In addition to the language itself, a complete business integration solution also requires: standardize tags, or metadata, for each commerce community; a means for mapping between different metadata descriptions; and a server for processing XML documents and invoking appropriate applications and services (Glushko, 1999). Integration of enterprises is mostly based on Electronic Data Interchange or EDI. However, EDI applications have proven to be expensive, and rigid. On the other hand, XML could provide the same benefits of EDI (improved efficiency, supply chain management, real-time data metrics, control systems, etc.) by dynamically adapting each process to the different clients applications. In addition, each of the elements of an XML's tree hierarchy has a specific relation, and navigation through the hierarchy is based upon this relation.

ERP systems and MES manage and capture a great amount of diverse information. However, this information should not be available to everyone in the same format, because of the use each department and person would give to the information. The data stored in a MES system should change in real-time, while the information needed by front offices (and ERP systems) may not require real-time information update. Therefore, XML provides a medium to move data across dissimilar systems with quick transformations of the data. XML maintains a separate interpretation from the user interface and the structured data allowing data to be integrated from different sources. The middle tier of XML architecture provides the channel to exchange abstract data online (Goldfarb, 2000).

XML presents many advantages because it separates data from applications, making it possible to reuse the same applications with different data and the same data with different applications. Since data interchange problems arise because of the difference in language, platforms and applications, XML could provide a solution for inter-enterprise communications.

4. Relation between Information Systems and Enterprise Wide Data

An enterprise is an organization composed of different departments working towards a common goal. Each part of an enterprise is autonomous from the others. However, they utilize common information resources.

The need for enterprise integration has yet to be fully understood. Moreover, the role of Manufacturing Execution Systems in the context of an enterprise in integration has to be identified: one has to accept the dependence of various applications on certain data or information.

It is in the process of the information sharing when communication, thus integration, problems arise. Once we have identified and displayed the relationship between systems and data, we can proceed to implement a solution.

Taking this into account, the following tables identify data sources, and the systems that would make use of the data in one way or other. They present the relation between information systems and data within an enterprise. It is important to notice that this is not an exhaustive list of the data one can encounter in an enterprise, it is a representation of enterprise information in order to understand the need for a good information interchange among systems of an enterprise. Thus, it represents how the data is needed by the different systems, and thereby, confirming the great need for a robust information sharing process.

4.1 Information Systems and Enterprise Applications

Each of the following figures contains the functional parts (the IS) of an enterprise, and the data, applications or reports needed by them. The purpose of the figures is to represent the needs of the same information by the different information systems.

In at figure 4.1, the *master schedule* and its components is presented . In order to generate the master schedule for a certain production period, it is necessary to know the quantity of products required by the customer or the quantity of the forecasted. The Supply Chain Management (SCM) makes this information available, and the Manufacturing Execution System (MES) and the Controls System utilize it. The SCM gathers the data from customers while the MES uses this data to develop the schedule and passes the information to the Control System to program the machines.

The *quantity of products required to be purchase* is an information gathered and used by the SCM. It is generated from customer and forecasting information and it is used to determine suppliers and purchase requirements. As with the *products to produce*, this information is provided by customer demand and forecasting. Depending on the product and the process, different parts are needed. The Process and Product Engineering (P&PE) would give the specifications on the product to purchase, while the MES would give the quantity and the time for when the components are needed.

The *Bill of Materials (BOM)* describes each of the components for each of the products produced by the enterprise. P&PE originates the BOM when the product is designed. The BOM is needed by the SCM in order to find suppliers for each part. MES needs it to allocate the processes for each part, to know how many of each part is needed and when should the part enter the process. Control Systems needs to know the specifications of each part to monitor and control the processes.

Figure 4.5 presents an example of a data that is needed by each of the six IS of an enterprise. This is the product code. Product code is an identification code for each of the products of the enterprise. It is used whether the enterprise is trying to sell the product, find a customer or a supplier for a specific part or product by the SCM. If service or warranty information on the product were needed, the SSM would use the product code. P&PE needs the information to be able to design or redesign the product, MES to schedule the production, Controls to monitor the quality and process, and ERP to determine the cost and price of the product.

	Information System					
	Enterprise Resource Planning	Supply Chain Management	Sales and Service Management	Process and Product Engineering	Manufacturing Execution Systems	Controls
Planning Systems						
Master Schedule: When to begin production and how much capacity is needed						
Quantity of products required to produce by specific part number						
Quantity of products required to purchase by specific part number						
Bill of material for each product						
Material manufacturing process routing steps						
Quantity of material required by part number						
Current inventory						
Lead time for each part number						

Figure 4.1: Master Schedule

	Information System					
	Enterprise Resource Planning	Supply Chain Management	Sales and Service Management	Process and Product Engineering	Manufacturing Execution Systems	Controls
Materials Requirements Planning: Determines when these items must be ordered or when to begin manufacturing them						
Quantity of products required to produce by specific part number						
Quantity of products required to purchased by specific part number						
Current inventory						
Lead time for each part number						
Time it takes to make products or order products						

Figure 4.2: Material Requirements Planning

	Information System					
	Enterprise Resource Planning	Supply Chain Management	Sales and Service Management	Process and Product Engineering	Manufacturing Execution Systems	Controls
Capacity Requirements Planning: How much time per operation is needed to meet the production schedule						
Quantity of products required to produce by specific part number						
Capacity of the resources (items per time unit)						
Product or part number						
Schedule						
Quantity of products require to produce by specific part number						
List of resources with identification number						
Work that has been produced						
List of work yet to be done						
Work in process						
Material plans requests						

Figure 4.3: Capacity Requirements Planning

	Information System					
	Enterprise Resource Planning	Supply Chain Management	Sales and Service Management	Process and Product Engineering	Manufacturing Execution Systems	Controls
Purchasing: purchasing requirements						
Material requirements						
Current inventory						
Quantity of products require to purchase by specific part number						
Raw material to be purchased						

Figure 4.4: Resource Acquisition and Purchasing

	Information System					
	Enterprise Resource Planning	Supply Chain Management	Sales and Service Management	Process and Product Engineering	Manufacturing Execution Systems	Controls
Inventory						
Current inventory (beginning quantities)						
Quantities consumed for each part number						
Material to be purchased						
Receipt date						
Supplier data						
Lot number						
Specific location of item in warehouse						
Bill of material of each product						

Figure 4.5: Inventory Management

	Information System					
	Enterprise Resource Planning	Supply Chain Management	Sales and Service Management	Process and Product Engineering	Manufacturing Execution Systems	Controls
Product data: for each product						
Bill of material Routing standards						
Machines and operations						
Machine set-up times						
Part configurations						
Tool data						
Flow and process chart						
Product code						

Figure 4.6: Product/Engineering data

	Information System					
	Enterprise Resource Planning	Supply Chain Management	Sales and Service Management	Process and Product Engineering	Manufacturing Execution Systems	Controls
Control Systems						
Process control parameters						
Quality control standards						
Process plan						
Product information						
Resources						
Current inventory (beginning quantities)						
Quantities consumed for each part number						
Materials to be purchased						
Cost of material						
Quantities of product required to produce by specific part number						
Cost of each product						
Cost of resources						
Resources utilized for production						

Figure 4.7: Process Control Systems

	Information System					
	Enterprise Resource Planning	Supply Chain Management	Sales and Service Management	Process and Product Engineering	Manufacturing Execution Systems	Controls
Human Resources						
Time and attendance						
Operator or workers information sheet						
Payroll processing						

Figure 4.8: Human Resources Management

	Information System					
	Enterprise Resource Planning	Supply Chain Management	Sales and Service Management	Process and Product Engineering	Manufacturing Execution Systems	Controls
Finance						
Accounts payable and receivable						
Order management						
Financial reporting						
Production schedule and dispatching						
Planning horizons in hours, days, or weeks						

Figure 4.9: Accounting and Finance

4.2 Information Systems and Purchase Information (Customer/Supplier perspective)

A purchase is an activity of the enterprise where the enterprise some times acts as the supplier, and some times acts as the customer. Taking this into account the two following figures represent the information needed for the enterprise to offer a purchase (the enterprise acts as the supplier, and the information is gathered for the customer). In the second figure, the enterprise is making the purchase (the enterprise is the customer, and the information is gathered for the supplier).

Whenever a customer is making a purchase, there is some information generated for the purchase request. The customer has to provide information about the payment, about itself and about the order. The enterprise would generate an order process and an invoice based on the information generated from within the enterprise.

At the end, to create the customer order the following information would have been needed: *Customer Information*: Name, address, phone number of the customer, etc. *Order Description:* The customer submits to the Sales and Service Management System (SSM) the product, quantity and date for delivery. From there, the P&PE of the enterprise generate the design and process plan. The process plan is developed by the P&PE and the MES, taking into consideration the information from the Control Systems. All along the order-manufacturing-delivery process, the SCM is involved. *Payment Information*: The costs for the purchase order are generated by the MES, the Human Resource Management department and the Finance department. And the ERP system would manage this information.

In the same way, when the enterprise is purchasing goods, the different IS take part in the process. Figure 4.11 presents the schema of the information needed to purchase goods. *Supplier information:* is managed by the SCM. It would contain name, phone, address and contact person for the supplier. *Purchase Information:* Here is the detailed information of the purchase. The date for delivery depends on the schedule of production. This information is made available by the MES with the master schedule. The product information is the description of the part being purchased. All specifications are generated by the P&PE when designing the product and are available through the bills of material (BOM). *Quote (price):* It has to be approved by the financial department. As said before, the financial applications are managed through the ERP

These two figures give a different perspective of how data from different information systems come together. It demonstrates that information systems could not perform alone. There is the need for data sharing between the information systems in order for an enterprise to perform its production activities.

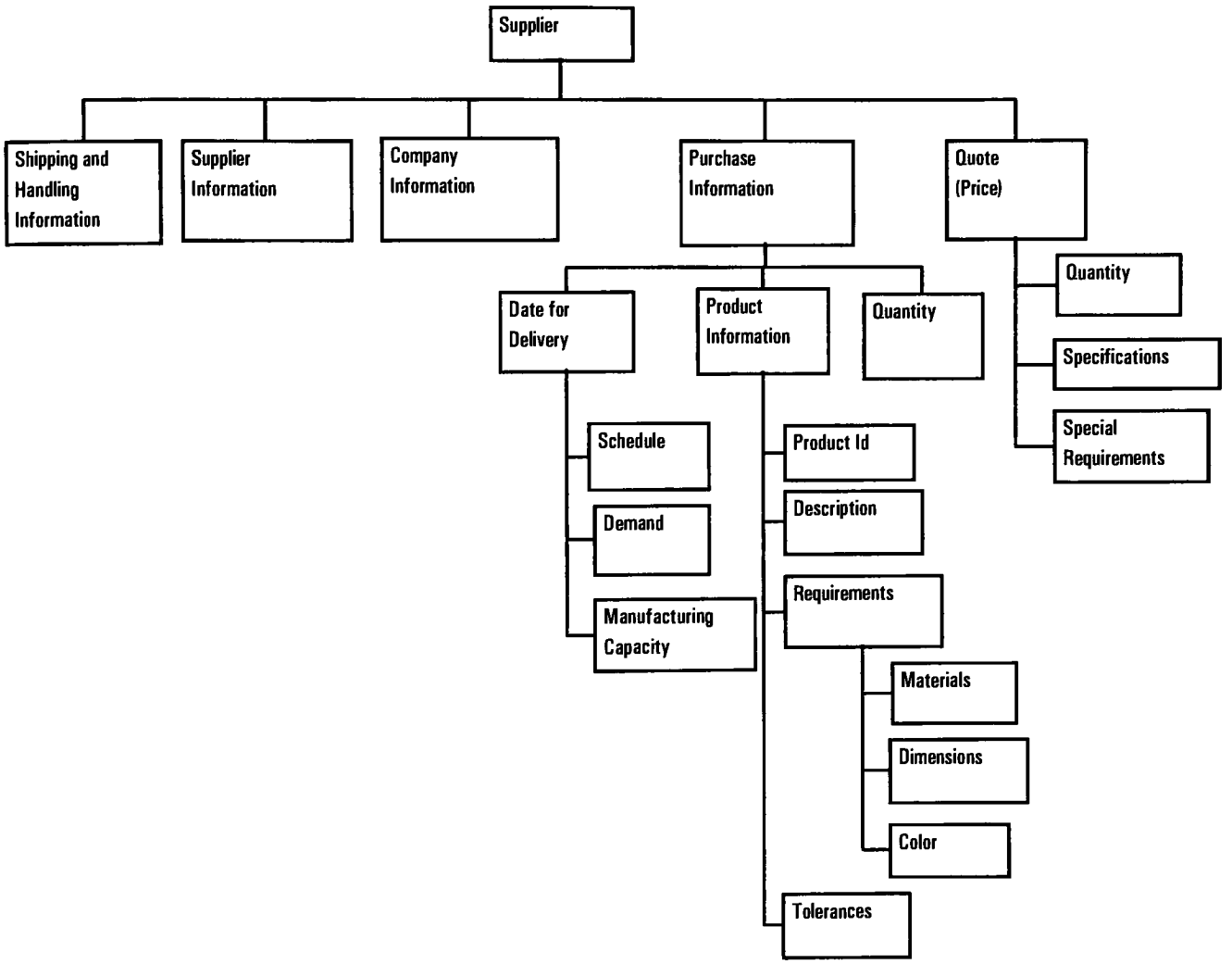


Figure 4.10 Supplier Perspective of Information Flow

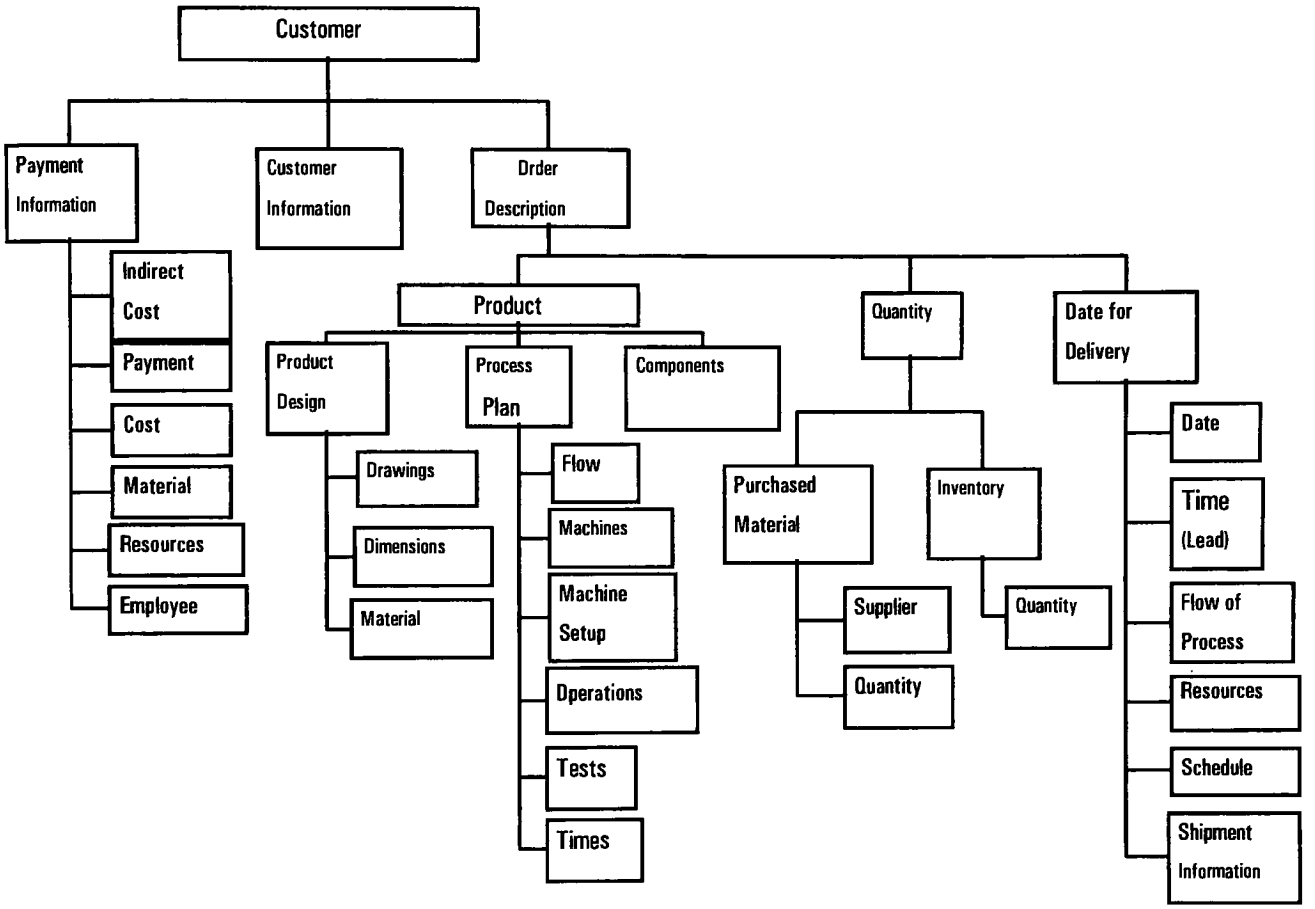


Figure 4.11 Customer Perspective of Information Flow

5. COMMOM OBJECT REQUESTED BROKER ARCHITECTURE

5.1 Distributed Computing and Software Architecture

Information and Computer Technology have been known to be fields with a fast evolution. However, the gross amount of IT industry's software products can't collaborate with one another, and consumers nowadays seek products that are re-usable, recyclable and with a long usage life.

Distributed computing is defined as the sharing of two or more pieces of software and their information which could be running on the same computer or on different computers connected to the same network (Otte, 1996). It can also be described as the allocation of the processing load among two or more computers (Linthicom, 1998). According to Otte (Otte, 1996) distributed computing is the ability to use computing resources more efficiently through:
- Sharing scare resources
- Sectioning out computing workload among different machines, and
  Assigning applications on the machines most appropriate for the application's needs.

For a system to implement distributed computing technology, the different computers have to communicate with each other, and the different software must also interchange information. To achieve reliable communication (better distributed computing) good software architecture should be elaborate. Software architecture is a conceptual model of how a system's components are defined and developed, and how they inter-operate (Ben-Shaul, 1998). In other words, it defines how the single parts of a computer system are going to interact when the system is running.

At the moment, different software applications generally don't communicate with each other, and if they do, the information shared is minimal and complicated to establish. Nevertheless, the need for integration of systems is growing. Computer Integrated Manufacturing systems are based upon distributed functions. Network Management Systems Architecture require information sharing in order to manage large networks and to handle changes quickly, and even between PC's applications, users encounter the need to insert spreadsheets into word processing documents, etc (Seetharaman, 1998, Murphy, 1998, Orfali, 1998, Haggerty, 1998).

Many answers have been developed as a result of this need, one is the Object Oriented Distributed Applications. Object oriented architecture consist of a set of rules, guidelines, interfaces and conventions used to define how applications communicate and inter operate with one another (Mowbray, 1997). The main object oriented models are CORBA, Microsoft's DCOM, STEP, and Sun Microsystems' RMI.

5.2 CORBA

The Object Management Group -OMG- has representations from IBM, IONA Technologies, Hewlett-Packard Co., Netscape, Sunsoft and many others, and since 1989 these companies have been developing standards and technology for object oriented distributed applications. The

standards are compiled as the Common Object Request Broker Architecture -CORBA- (Seetharaman, 1998; Orfali, 1997; Otte, 1996).

Murphy defines CORBA as a protocol for communicating among heterogeneous systems (Murphy, 1998). But others authors see CORBA as a software (or object) bus, where applications are brought together to allow information sharing with one another (Seetharaman, 1998; Orfali, 1997). CORBA gathers the specifications as how to develop object oriented technology, so that every software or computer application would be capable of sharing information.

CORBA establishes the interaction of components independent of the designers, the platform they're running on, the language they are written in, and where they're executed (Seetharaman, 1998; Siegel, 1998). This can be done because CORBA defines objects as a unifying metaphor for bringing existing applications to the "bus". It also denotes how once they are in the bus the intelligent components, the objects, are able to discover each other and inter-operate by means of bus-related services which include creating and deleting objects, accessing them by name, storing them in persistent stores and some others we'll approach later. (Orfaly, 1997). Each one of these objects' characteristics is specified in CORBA.

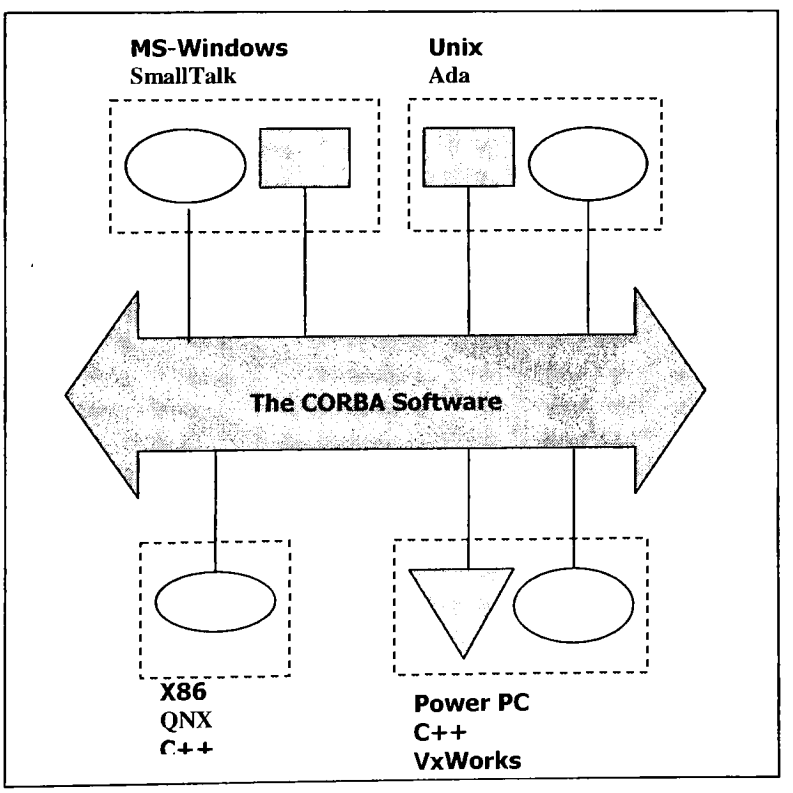


Figure 5.1 (Murphy, 1998) CORBA allows for interoperability among components, without concern for OS, programming languages, platforms or location.

Orfali explains that the CORBA object bus defines the shape of the components that live within it and how they interact (Orfali, 1997). By this CORBA allows systems to use different hardware, operating systems, and programming languages to communicate (Murphy, 1997), and it complements the use of distributed computing by:

Allowing a flexible, changing relationship between applications
Adding an intermediary (to the communication): a broker
Allowing applications to handle more than single processes
Supporting both synchronous and asynchronous communication styles
(Otte, 1996).

With CORBA specifications, the application integration race has found a supporter. The IDL language and the ORB interface, both main parts of CORBA, contribute to the development of good software architecture, and by adopting CORBA, the computer industry is choosing to create an open interaction field for components (Mowbray, 1997, Orfali, 1997).

5.2.1 CORBA COMPONENTS

The pillars where the CORBA Object Oriented Architecture is supported on are the Object Request Broker (ORB) and the Interface Definition Language (IDL). The ORB is a middleware component that acts as an intermediate between objects which make requests to and receive responses from other objects (Orfali, 1997). IDL is necessary to code information of an object into a set of interface definitions (Otte, 1996). This interface definition specifies the operations the object is prepared to perform, the input and output parameter each requires, and any exceptions that may be generated along the way (Siegel, 1998).

Beside these main, and crucial, CORBA elements other components are necessary in an object-oriented architecture developed with the CORBA specifications.

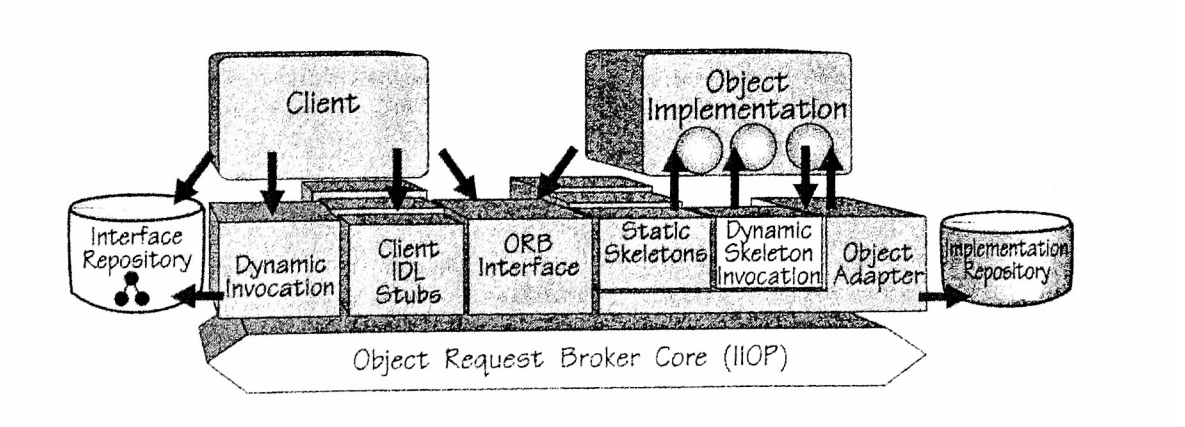


Figure 5. 2 (Orfali, 1997). Structure of a CORBA ORB.

- The servant: allocated in the server side, or server object, it implements the operations defined by an IDL interface. It's identified by its object reference (Schmidt, 1998).

- Object reference: uniquely identifies the servant object in a server process.

- The client: a program whose application tasks are executed by obtaining objects references and invoking operations on servers. The server objects can be remote or local relative to the client.

- The ORB core: delivers the request to the servant and returns a response to the client. An ORB core is typically implemented as a run-time library linked into client and server applications (Schmidt, 1998).

- The ORB interface: is a logical entity that may be implemented in various ways, as one or more processes or a set of libraries. The CORBA specification defines an abstract interface for an ORB, which may be implemented in different ways. This ORB interface provides standard operations that: initialize and shut down the ORB, convert object references to string and back, and creates argument lists or requests made trough the Dynamic Invocation Interface (Schmidt, 1998).

- OMG IDL Stubs: allocated on the client side, they provide a Static Invocation Interface (SII) that arranges application data into common packet-level representation. IDL stubs define how clients invoke corresponding services to the server. It's like a local call, a local proxy for a remote server object (Orfali, 1997; Schmidt, 1998).

- OMG IDL Skeletons: they convert back the packet-level representation into typed data that is meaningful to an application. They are the server's side equivalent to the OMG IDL Stubs. Both, IDL stubs and skeletons, are generated by the IDL compiler (Orfali, 1997; Schmidt, 1998).

- IDL Compiler: Automatically transforms OMG IDL definitions into application programming language (C, C++, Java and others). The benefits of the IDL compiler: provides language transparency, eliminates common sources of network programming errors, and provides opportunities for automated compiler optimization (Schmidt, 1998).

- Dynamic Invocation Interface (DII): allows clients to generate requests at run-time. Also permits clients to make deferred synchronous calls, which separate the request and response portions of two-way operations to avoid blocking the client until the servant responds. DII provides high flexibility, but requires more coding than IDL-compiled invocations (Schmidt, 1998; Siegel, 1998).

- Dynamic Skeleton Interface (DSI): Enables an ORB to deliver requests to a servant that does not have IDL-based compiled skeletons. The servant doesn't know the IDL interface it's implementing. The DSI looks at parameter values in an incoming message to figure out the target and method (Orfali, 1997; Schmidt, 1998).

- Object Adapter: Provides the run-time environment for the interaction between the ORB and the server object. It links a servant with an ORB, demultiplexes incoming requests to the servant, and sends the appropriate operation upcall on that servant. Also, object adapters make it possible for an ORB to support various types of servants that posses similar requirement (Orfali, 1997; Schmidt, 1998).

- The Interface Repository API: Is a run-time source of information about IDL interfaces. The API allows components to dynamically access, store, and update metadata information of all interfaces in the system. This permits every component in the ORB to have self-describing interfaces (Orfali, 1997).

- Implementation Repository: Provides information about the server (the object id, the servants it contains, and the classes it supports), that allows the ORB to locate and activate the server object (Orfali, 1997; Schmidt, 1998). Interface repositories (with unique identifiers and names), assure access to IDL interface definitions to all ORBs (Siegel, 1998).

5.3 OBJECT REQUEST BROKER

5.3.1 Definition and how it works

CORBA was conceived to be developed using objects. A CORBA object models a real-world object and consist of data and methods that may be invoked on it (Seetharaman, 1998). These objects are packaged as binary components that remote clients can access via method invocations, and because of this object infrastructure, it's easier for components to be more autonomous, self-managing and collaborative (Orfali, 1997).

Because CORBA's goal is interoperability independent of designer, platforms, language, and location, the server object's language and compiler have to be transparent to clients (Seetharaman, 1998; Siegel, 1998). In the same way, location and operation system of the object are not important issues for the client (Orfali, 1997). This client side transparency also allows the developer to make changes to the application implementation without affecting the clients using the implementations or services (Mowbray, 1997).

The Object Request Broker (ORB) is software that acts as an intermediate between the client and the server, it manages the client requests and hides the location and implementation details of the server from the client (Seetharaman, 1998). The ORB is that object bus where requests are made and responses received from objects located locally or remotely (Orfali, 1997).

The invocation process occurs as follows: a client's object making a request will invoke a method on a server. The ORB intercepts the call and is responsible for finding a server object that can implement that request, pass it the parameters, invoke its method, and return the results to the client. The ORB is the reason why the client does not have to be aware of where the object is located, its programming language, its operating system or any other system aspect that are not of an object's interface (Orfali, 1997). All the requests do not pass directly from client to object implementation, requests are always managed by an ORB (Siegel, 1998).

There are two ways clients can make their requests (Siegel, 1998):

1. Static invocations: the requests are routed to the ORB through a client stub compiled in the server's programming language from the IDL interface definition.
2. Dynamic invocations: these requests are set up at run time by the client using a set of ORB functions defined for this purpose.

An ORB supports the use of a flexible architecture by allowing the dynamic binding of a component's client invocation to the actual implementation at run-time (Mowbray, 1997)

Besides managing communications between clients and servers, ORBs communicate with each other. Communication between ORBs is the basis of CORBA distribution. With General Internet Inter-ORB (GIIO) and Internet Inter ORB protocol (IIOP), ORBs can send to each other standard messages, letting clients and objects see object-oriented invocations and responses. OMG IDL supports distribution in different ways: it enforces encapsulation and defines operation types (Siegel, 1998).

Orfali (Orfali, 1997) lists some benefits provides by CORBA's ORB:

- Self describing system: CORBA's ORB supports an Interface Repository, which contains real time information about the functions a server provides and their parameters.

- Local/remote transparency: An ORB can act as a broker between objects that resides within a single process, multiple processes running in the same machine, or multiple processes running across networks and operating systems.

- Built in security and transactions: the ORB includes information in its messages to handle security and transactions across machines and ORBs boundaries.

- Polymorphic messaging: ORBs invokes a function on a target object. Depending on the object that receives the call, the same function call will have different effects.

- Coexistence with existing systems: New applications can be written as pure objects and existing applications can be encapsulated with IDL wrappers.

- Static and dynamic method invocations: A CORBA ORB either lets you statically define your method invocations at compile time, or it lets you dynamically discover them at run time.

- High level language bindings: CORBA separates interface from implementation and provides language-neutral data types, making it possible for ORBs to call objects across language and operating system boundaries

5.3.2 Interoperable Object Reference –IOR-

The Interoperable Object Reference (IOR) is the representation, for the client, of an object's specification (Siegel, 1998). Object references uniquely identify local or remote objects in order to differentiate objects from each other. Clients can only request an operation on an object if they hold a reference to that object (Henning, 1998).

The format of the object reference is an OMG standard that allows interoperability, but it's opaque to the client, to provide transparency to applications such as language, transport and protocol independence (Siegel, 1998; Henning, 1998). Saying that the object reference is opaque, really means that application code is not allowed to work inside a reference, or to make any assumptions about its internals (Henning, 1998).

An IOR contains three types of information about an object:

1. Type name: the type name, or repository id, identifies the most derived type of the object associated with the IOR. The repository id serves like an index into the Interface Repository (IFR). The interface repository helps the IDL definition to retrieve an interface during run time.

2. Protocol and address details: Specifies a protocol and the address information for that protocol.

3. Object key: the object key identifies which particular object is pointed to by the IOR. The object key consists of two components: object adapter name, which identifies the object adapter in a server that accepts request for a particular object; and object name, which identifies what object within the adapter is pointed to by the IOR.

CORBA specifications allows a single reference to contain several pairs of protocol and object key, so that a single object reference can support multiple protocols or have different addresses for the same object.

CORBA defines different types of references, one for each of the two types of CORBA object the Object Adapter supports, transient and persistent (Vinoski, 1998).

Transient References: An IOR is transient when it works only as long as its associated server process remains available. Binding of transient references needs the server to be available at the time a client requests an operation. Henning (Henning , 1998) details the process as follow:

1. The client-side ORB run time opens a connection to the host and port specified in the IOR, and sends a request message to the server process. A request message contains: the size of the message, a request ID, the object key, the name of the operation being call and any in and inout parameters that need to be transmitted to the operation.

2. With the adapter name and object name, the server locates the correct servant for the request.

3. Finally, the server sends a reply message to the client. The reply message contains the request id sent initially to the server by the client, this allows the client to locate a request with its reply, so it can have several requests outstanding.

Persistent References: A persistent IOR holds reference to the same object despite the server shutting down and restarting in the same machine or in another machine. Persistent objects can live beyond any particular process in which they are created or activated (Vinoski, 1998).

A server could start on different hosts at different times, or the server port number could be dynamically assign, and a persistent IOR is expected to work even though the port or host have change. The Implementation Repository helps bind persistent IORs, The Implementation Repository maintains a registry of known servers, records which server is currently running at which host (or computer) and port number, and starts servers on demand if they are registered for automatic activation (Henning, 1998).

Binding of persistent objects is not very different from binding of transient ones, at least for the client side. Persistent references contain the repository id of the most used interfaces, the object key (with its two components object adapter name and object name), the host name and the port number of the implementation repository.

According to Henning the process is as follows: the client opens a connection to the address in the IOR and sends the request. This connection leads to the implementation repository and retrieves the object adapter name to use it as an index into its server table. The rest of the process is the same as with transient IOR (Henning, 1998).

Because of this needed contact with the Implementation Repository by the client, problems emerge: a great amount of requests have to be retransmitted using the same repository, but repositories can only handle a limited quantity of servers. In order to solve this problem, CORBA suggest the following solutions:

> Instead of operation request, the client-side run time could send a locate request to the repository. This is worthwhile only for large requests.
> More than one implementation repository could be used, reducing the size of each location domain, thus reducing the request send to the repository (Henning, 1998).

5.3.3 Fault Tolerance, Object migration, and Garbage collection (Henning, 1998)

Fault tolerance and load sharing among IOR are important concepts in order to deliver better performance. This can be achieved because an ORB can embed multiple addresses in an IOR, each of which indicates a redundant reference of the same repository. Also, reducing the size of location domains improves fault tolerance, for example if each host runs its own repository. Nevertheless, reducing size of location limits object migration.

Object migration is the ability to move an object from one address space to another without breaking the references to the object held by clients. Ideally, an ORB would permit free

movement of objects, but for this to be possible the implementation repository needs the information of objects (not servers) which presents a storing problem because of the large number of objects. A way to approach this problem is if the repository communicates with the server binding and dynamically obtains the location information of the object. This process requires more messages and reduces performance.

Servers could also migrate across location domains, (even though currently no commercial implementations offer interdomain migration), but with the same problem as object migration across servers, migration across domains sacrifices performance and scalability. There are two possible solutions:

1. If a server migrates to a new domain, it registers itself with a new Implementation Repository, and also all repositories it had used in the past.
2. If a server migrates to a new domain, an administrator changes the registration in the old repository to create location-forward replies to the new repository. This alternative has a draw back, the forwarded messages accumulate over time.

CORBA does not contemplate any process for deleting information not longer require by clients, like old forwarded messages. CORBA lets persistent references propagate by uncontrollable means, therefore leaving no way of knowing if an object is of interest to a client. The most common method for destroying objects is to determine if they have not been accessed for some time.

Deleting objects is a very critical operation. For example, if a client decides to delete an object while some other clients still hold references to that object, these references hang loose. A CORBA system needs every object to have a reference, and that every reference belongs to an object, this is called referential integrity. A garbage collection feature could prevent the inadequate deletion of objects and guarantee the deletion of unwanted objects. Unfortunately, CORBA does not consider garbage collection at this point.

5.3.4 Multithreaded ORB:

Multithreading is an architecture technique that allows operations to be carried out simultaneously without disrupting the progress of other operations. ORBs can be developed using multithreaded architecture in order to easily service multiple concurrent requests (Vinoski, 1998). Schmidt list some benefits of multithreaded ORBs (Schmidt, 1998):

- With multiple threads, each request can be served in its own thread, independent of other requests.

- Systems resources are maintained, since creating a new thread is typically less expensive than creating an entirely new process.

- Program design is simplified by allowing multiple servants to execute requests independently using conventional programming abstractions. For example CORBA's synchronous remote method of requests and replies.

- Improvement of end-to-end throughput and latency performance by using the parallel processing capabilities of multiprocessor hardware platforms and by overlapping computation with communications.

- Improvement of perceived response time for interactive client applications by associating separate threads with different operations in order to avoid clients operations to block indefinitely.

Different multithreading architecture for ORB

There are different approaches to multithreading architecture, each with its own characteristics. In order to use the best suited thread strategy in an application, a number of factors have to be consider: number of objects hosted by the application, the expected request rate, and the multithreading support provided by the underlying operating system (Vinoski, 1998). The following multithreading architectures were described by Schmidt (Schmidt, 1998) and are focused on the server side.

1. **Thread per request**: each request from a client is handle in a separate thread.

Advantages: -It is straightforward to implement.
-Useful to handle long duration requests.

Disadvantages: -It can use a large number of OS resources if many clients make requests simultaneously.
-It is inefficient for short-duration requests, this is because of excessive thread creation overhead.
-It's not appropriate for real time application, because the cost of creating a thread for each request can be non-deterministic.

2. **Thread per connection**: each thread is dedicated to handle a separate client for as long as it's connected.

Advantages: -It is straightforward to implement.
-It's convenient for ORBs that perform long-duration conversations with multiple clients.

Disadvantages: -Does not supports load balancing effectively.

3. **Thread per servant**: associates a thread for each servant registered in the ORB's object adapter.

Advantages: -Minimize the amount of rework required to multithread existing single-threaded servants.

Disadvantages: -Does not supports load balancing effectively.

4. **Thread pool architecture**: This is a variation of the thread per request. It minimizes thread creation cost by pre-defining a pool of threads. With thread-pool architecture, client requests can be executed concurrently until the number of simultaneous requests exceeds the number of threads in the pool. At this point, requests wait in a queue for the next available thread.

Thread-pool architecture has different approaches:

- *Worker thread-pool architecture*: It includes an I/O thread, a request queue, and a pool of worker threads.

Advantages: -Ease of implementation

Disadvantages: -Excessive context switching and synchronization required to manage the request queue, and the request-level priority inversion because of connection multiplexing.

- *The leader/follower thread-pool architecture*: a pool of thread is allocated and a leader thread selected on connections for all servants in the server process. When a request arrives the thread reads it, and if it is a valid request for a servant, a follower thread in the pool is released to become the new leader and the leader transmit the request. Then the original leader thread becomes a follower back at the thread pool.

Advantages: -Minimizes context switching overhead incurred by incoming request.

Disadvantages: -The disadvantages are the same as worker thread-pool architecture.
-Also, it's harder to implement.

5. **Threading framework**: It provides methods to application developers for customizing ORB architecture.

Advantages: Flexibility, it can support various strategies.

Disadvantages: -It is generality can significantly increase locking overhead.

6. **Reactor per thread priority architecture**: This architecture is based on the reactor pattern. It integrates transport endpoint demultiplexing and the dispatching of the corresponding event handlers. This threading architecture associates a group of reactors with a group of threads running at different priorities.

Advantages: Minimizes priority inversion and nondeterminism.
-Reduces context switching and synchronization overhead
-Supports scheduling and analysis techniques that associate priority with rate.

Disadvantages: It serializes all client requests for each reactor within a single thread of control

Multithreaded ORBs can improve the efficiency and predictability of client and server applications. Also, they can yield simpler servant implementations than single-threaded ORBs (Schmidt, 1998).

5.3.5 Remote Procedure Calls vs. Object Request Broker

Remote Procedure Call (RPC) is a technology which was the base of the first attempt of CORBA, because it had already been used in production of distributed systems (Vinoski, 1998). The RPC protocol is a procedure that allows a user in a computer to invoke a function on another computer (Murphy, 1998). With a remote procedure call, the call is made to a specific function, where the data is separated from the function (Orfali, 1997).

The ORB substituted RPC in later CORBA versions. With ORB the call is made to a method within a specific object, and different object classes may respond to the same method invocation in different ways. This is because each object manages its own private data and the method is implemented on that specific instance data. But with RPC calls all the functions that have the same name get implemented the same way (Orfali, 1997).

The difference between RPC and ORB is that ORB architecture has the ability to specify an object without necessarily specifying the server that contains the object. That location independence allows references to objects to be passed freely around the network (Murphy, 1998).

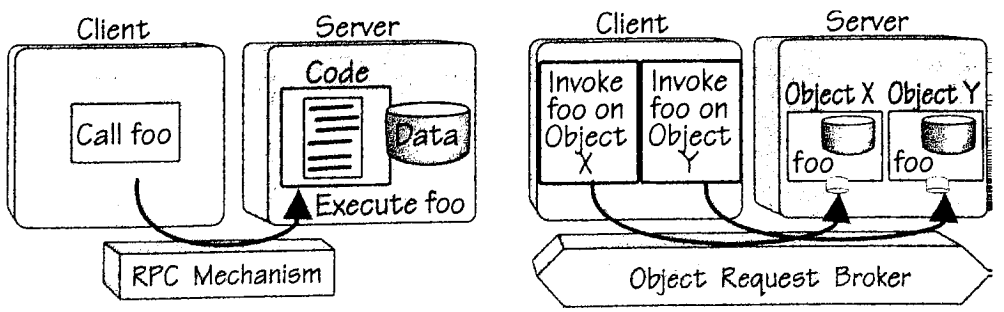


Figure 5.3 (Orfali, 1997). On the server side of an ORB implementation each object contains its own data and the method implemented on that data.

5.4 Interface Definition Language - IDL -

CORBA's interface definition fully describes an object, the operations supported by it, the input and output parameters required by the operations and any other characteristic or behavior particular of the object (Otte, 1996; Siegel, 1998). In order to define interfaces under the CORBA principles, the OMG has established a standard Interface Definition Language (IDL). With IDL developers can define interfaces and data structures that can later be retrieve using high level languages (Otte, 1996; Siegel, 1998).

CORBA objects can be placed on one location and accessed from another. To accomplish this the object has to be defined as language and platform independent, and that's the function of the IDL (Murphy, 1998). It provides an implementation-independent language for specific interfaces, and IDL can be compiled and implemented from number of specific languages (Orfali, 1997).

As Siegel (Siegel, 1998) explains, the OMG IDL compiler uses mapping specifications to generate a set of method invocations from the IDL operations. Developers refer to the OMG IDL file (this contains the definitions of interfaces the client or server implementation will support) and use the language mappings to generate the corresponding set of method declarations. After compilation the skeleton is ready to make the right calls to invoke operations on the object implementation.

OMG IDL is used to define interfaces and data structures, but not to write algorithms. IDL is use to generate source code for the desired programming languages. At this moment IDL has language-mapping specifications for C, C++, Java, Smalltalk, and Ada. But the greatest advantage of using IDL is that it frees CORBA applications from being restricted to any particular programming language (Mowbray, 1997; Orfali, 1997; Otte, 1996; Siegel, 1998).

IDL supports location transparency, this means that a remote distributed object can't be distinguish in its invocation from a local component. Also, IDL keeps the system unaware of changes in the implementation of the software components and the physical allocation of components until run-time. This is also a feature carry out by ORBs (Mowbray, 1997).

Even though IDL isn't the only way to distribute a system into useful components, it is relatively simple and provides support for both industry and standard organizations (Mowbray, 1997).
See "Appendix" for code examples

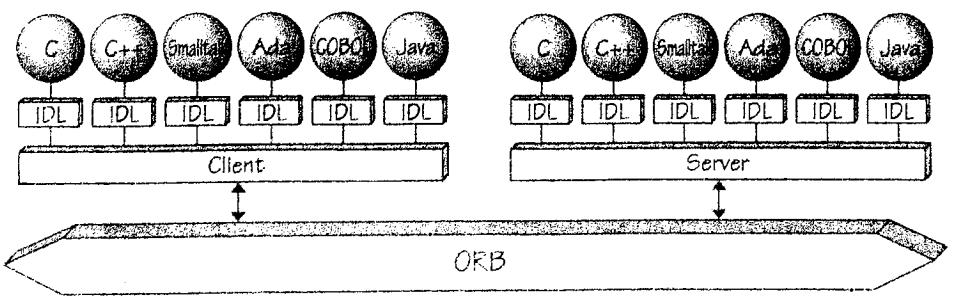


Figure 5.4 (Orfali, 1997). CORBA IDL Language Bindings provide client/server interoperability.

5.5 Object Management Architecture – OMA -

The Object Management Architecture (OMA) is a service-based architecture. In other words the access of distributed objects can only be done through their interfaces, without having knowledge about the implementation details that support a particular interface. The benefit of this service architecture is that servers are detached from the clients that request them. By being independent, servers can be shared among multiple clients and this allows a service to be reused within the distributed environment (Mowbray, 1997).

What the OMA pursues is a standard interface for the applications that provides basic functionality. It makes those applications reusable and widely available and thus minimizes the amount of new code needed for an application (Mowbray, 1997; Siegle, 1998).

The OMA defines four groups of standard applications and the functions each have. These applications are independent from one another and ORB software needs the support of these applications to be able to maintain a stable distributed object environment.

The CORBA Architecture can be represented as follows:

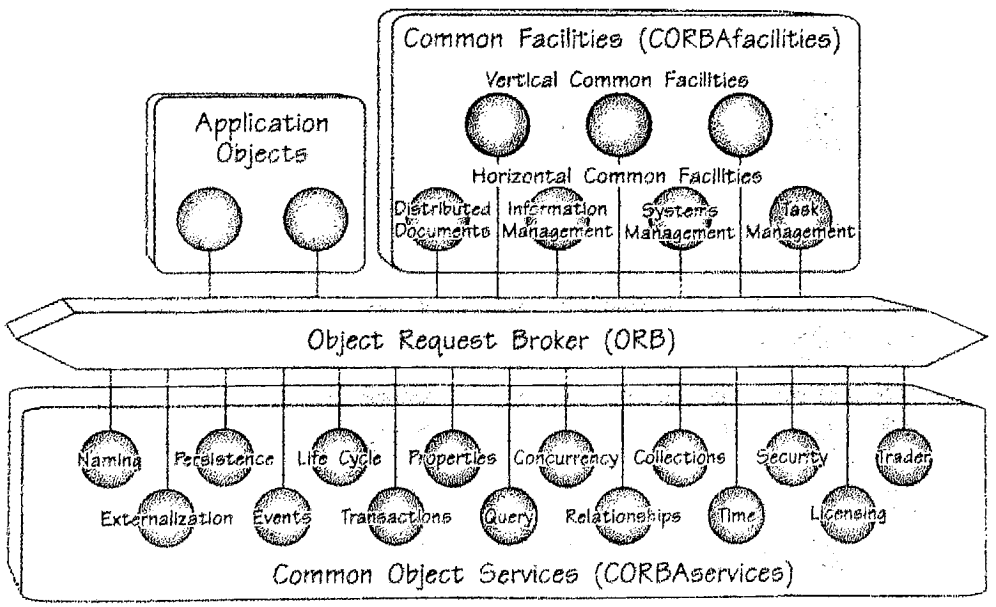


Figure 5.5 (Orfali, 1997). The Object Management Architecture.

5.5.1 CORBA Services

CORBAservices defines services needed to: create objects, control access to objects, track objects and their references, and also defines other functions that complement and augment the operability of the ORB (Otte, 1996; Orfali, 1997). These services are standardized by the OMG

to facilitate the development of applications and high level services across CORBA implementations (Mowbray, 1997).

Up to now there are fifteen services, and even though vendors do not have to provide every service, the ones they do provide have to comply with the OMG and OMA specifications (Siegle, 1998).

1. Life Cycle service: A client can create a new object using the operations defined in the life cycle service. Clients can create, copy, move or delete an object.

2. Persistence service: Distributed objects must maintain their state even if the program that created them terminates. The persistence service provides a "storage place" with a single interface for a variety of storage servers like object databases, relational databases and simple files.

3. Naming service: The naming service makes possible to locate an object if the name is known. This service provides the ability to link a name to an object reference; by giving the object name, you can retrieve the object reference and contact the object.

4. Event service: An event is an action occurring in an object that could be of interest to one or more objects. The event service allows components to notify their interest in specific events, and the occurrence of events. This service defines an object called an Event Channel, which collects and distributes events among objects unknown to each other. Haggesty defines the event service as a service that supports decoupled event delivery from an event supplier to an event consumer (Haggesty, 1998).

5. Concurrency Control service: Objects can coordinate the access to shared locations using locks. This service provides a lock manager that can obtain locks for transactions or threads (Orfali, 1997).

6. Transaction service: Transaction becomes the fundamental unit of recovery, consistency and concurrency in a distributed object system. The transaction service provides a two-phase commit and rollback coordination for recoverable components using flat or nested transactions (Orfali, 1997; Siegel, 1998).

7. Relationship service: A relationship is defined by a set of roles that two or more objects play. The relationship service allows components and objects that know nothing of each other to be related. It provides a way to create dynamic relationships between immutable objects.

8. Externalization service: Provides a standard way of getting an object's content into a stream, so it can be exchanged or moved around.

9. Query service: Allows the user to find objects whose attributes meet the search criteria specified using a query.

10. Licensing service: With this service the use of components can be meter to guarantee a fair compensation. The component informs the service when it's first used, when it is still in use, and when it stops being used. Any model of usage control at any point in a component life cycle is supported, like charging per session, per node, per instance creation, and per site.

11. Property service: Any object can be associated to named values or properties. The users can dynamically associate a title or date, for example, with a component's state.

12. Time service: It has five main functions:

    a. Synchronizes time in a distributed-object environment.
    b. Obtains current time with an estimate of error.
    c. Establishes the order in which events occur.
    d. Generates time-based events based on time and alarms
    e. Computes the interval of time between two events.

13. Security service: This service provides the structure for distributed object security. It supports authentication, access control, and confidentiality. An authenticated id makes clients accountable for their actions, restricts access to servers, and identifies the originator of a message. Because objects are free to move around, security is essential in areas such as business, commerce, and defense (Siegel, 1996).

14. Trader services: It associates an object with the service they provide. This service is useful when the user doesn't know which object in the system will be useful for a certain task.

15. CORBA Collection service: Provides standard interfaces to create and manipulate groups of objects.

5.5.2 CORBA Facilities

The Common Facilities are specifications developed for general-purpose application capabilities (Otte, 1996). The CORBAfacilities can handle application-level information, which could determine the user's view of the data.

The CORBAfacilities are developed by OMG as specification for general-purpose application capabilities that handle application-level information (Siegel, 1996). For example, facilities for compound document management, access of databases, a meeting schedule, or synchronizing times in a distributed system.

At this time OMG has defined the Horizontal CORBAfacilities and the Vertical CORBAfacilities or (CORBAdomains). The Horizontal facilities include four major categories: **User Interface,** the presentation of compound documents; **Information Management**, the interchange of compound documents; **System Management**, this facility tries to extract and apply standards that will allow management tools to work together; And **Task Management**, includes workflow, agents, rule management and object automation (Siegel, 1996).

The Vertical facilities are standard services within a particular domain. Its technology that supports certain market segments, and whose intention is to provide a standard interface definition so that clients of a specific industry can choose implementations with minimal application changes (Mowbray, 1997).

Standards developed by an industry group can be adopted as a Vertical CORBAfacility (Siegel, 1996). The OMG helps independent organizations produce standards by forming a liaison relationship, Domain Task Forces, and helping the organization define the specifications in OMG IDL according to the OMA (Siegel, 1998).

The approval of Vertical CORBAfcilities has three steps according to Siegel (Siegel, 1998):

1. The Domain Task Forces, DTF, write requirements documents for new specification, and evaluate and recommend candidate specifications.
2. Every domain specification has to have the support of all domains beside the one where it was originated.
3. The approval of the specifications by OMG.

The different Domain Task Forces are Business Objects, Finance/Insurance, Electronic Commerce, Manufacturing, Health Care (CORBAMed), Telecommunications, Transportation and Life Science Research.

The completed specifications are:

A Currency Facility (Finance).
A set of Product Data Management Enablers (Manufacturing).
Person Identifier service (Health Care).
Lexicon Query service (Health Care).
An Audio/Visual Stream Control Object (Telecommunications).
The Notification Service (Telecommunications).

5.5.3 Application Objects

Application Objects are object-oriented applications, which perform specific tasks for the end user. The OMG does not want to standardize application objects, for it will be where the vendors will compete, bringing innovative ways to provide the best features and value for the customer (Otte, 1996; Siegel, 1998).

5.6 CORBA's New Features

Every technology has continually evolved since its original release, and CORBA isn't the exception. Whenever changes or additions are going to be made to the CORBA specifications, the OMG has a process of approval. For new specifications the OMG requires a document called Request for Proposal. Currently, the request for proposal ask for a component model for CORBA systems, structured as a natural extension of the existing CORBA object model (Haggesty, 1998).

The next CORBA version (version 3.0) will have significant new features. Vinoski (Vinoski, 1998) explain three of the most important:

1. The Portable Object Adapter (POA): Before conceiving the POA, OMG defined the Basic Object Adapter (BOA), which provided basic services for objects to be created and implemented.
   The POA is a mediator between ORB and the server application. It uses the object id to associate a target object and a programming language servant. The POA sends the invocation to the servant and receives back the response. Then it sends the response to the ORB, who then sends it to the client.
   The POA specifications allow a single CORBA object to be incarnated by one or more servants during its lifetime. But the main goal of the POA is to provide portability for different types of CORBA server applications.

2. CORBA messaging: The CORBA messaging specification includes time-independent invocation, asynchronous messaging and facilities for specifying messaging Quality of Service (QoS).
   But is the asynchronous messaging what makes this specification so important to OMG. Asynchronous messaging assumes that the connection between two systems won't be always reliable. Before asynchronous messaging, there were three models for request invocations:
   - Synchronous: the client invokes an operation and waits for the response.
   - Deferred synchronous: the client invokes an operation, continues processing and the goes back latter for the response.
   - One way: the client invokes an operation and the ORB provides a best-effort guarantee the request will be delivered. No response is send back to the client.

   The two new asynchronous models are:

   - Callback: the client sends the request and also an object reference parameter. The ORB uses the object reference to deliver the response back to the application.
   - Polling: when the client invokes an operation, it returns a value-type used to pool or wait for the response.

   Also, the CORBA messaging increases GIOP and IIOP ORB interoperability protocols, letting these protocols deliver requests to intermediate routings agents. These agents can provide "store and forward" capabilities that can keep messages if a target is not available or active.

3. Objects by value: CORBA specification allows references and data types to be passed to objects and operations as arguments, but recently support for passing objects by value was added.
   The value-type is a new component of OMG IDL. It is a cross between a struct and an interface, and supports both data members and operation.

All operations invoked on value-types are always local to the process in which the value-type exists, because value-type invocations never involve the transmission of requests and replies over the network.
Value-types can be defined in IDL and are best to represent lightweight encapsulated data types, i.e. graphs, lists, and dates.

5.7 Benefits of CORBA

The intention of CORBA is to create a better communication system among applications. The way its being developed allows for constant improvement and future growth of the system, and the distributed capabilities of CORBA helps manage large networks (Haggesty, 1998).

Paul Haggerty (Haggerty, 1998) explains the main benefits of implementing the CORBA standards in a network system:

1. CORBA's object is capable of handling and changing new features with minimum redesign and coding.

2. The distributed architecture can handle a high degree of data collection and monitoring.

3. The location of an object in the system is completely transparent, objects interact with each other if they're local or remote.

4. A fault event can be better handle in a distributed system, compared with a non-distributed system.

5. Because of the standard interface (the Interface Definition Language) management systems can manage network resources in a common way, and telecommunication networks can exchange management information using standard interfaces.

6. CORBA forces separation of an object interface and implementation from its use. With this, the application code and the user interface dependencies are kept apart.

7. CORBA is programming language independent. Different parts of a system can be written in different languages and still be integrated without much trouble.

8. The Internet Inter ORB Protocol (IIOP) provides support for Web access. With a Web browsers a system can have better accessibility, portability and simplicity in an Intranet environment.

9. CORBA interfaces facilitates the integration of applications into customers' applications, network management systems, and industry-standard network management system.

10. CORBA is always evolving. It is not a set standard, but rather a changing specification. But because of the object model it is based on, changes in CORBA specifications are easy to adopt and they benefit the distribution environment.

6. Extensible Markup Language

6.1 Standard Generalized Markup Language – SGML –

The Standard Generalized Markup Language is a tool for defining and implementing some aspects of text processing applications (Goldfarb, 1990). It isolates any visual information in a document by declaring the information in an explicit way (van Herwijnen, 1994). Moreover, SGML is a method of representing text electronically by describing only the formal properties and inter-relations of the components of a document (TEI). The function of SGML is to define the content of a document in terms that are entirely independent of its processing.

SMGL provides a general-purpose mechanism for string substitution. A simple machine independent way of stating that a particular string of characters in the document should be replaced by some other string when the document is processed. This way, documents can be sent from one system to another without loss of information or misinterpretation. The transferred document (which is independent of application-specific information) can be processed as required. SGML makes a document portable because it contains an unambiguous description of its logical structure, independent of its visual form (van Herwijnen, 1994).

Contrary to the general idea that SGML is a formatting language, SGML is decidedly unhelpful about how texts are to be reproduced, it places a text in electronic form. By separating the notion of what the text actually is from how the text is presented, it makes possible the use of the same text by many different kinds of processors.

SGML is text encoding and interchange, and markup is synonym of encoding. Markup language is a set of marked up conventions used together for encoding text, it is making explicit an interpretation of a text. It must specify what markup is allowed, what markup is required, how markup is to be distinguished from text, and what the markup means.

The function of the markup is to identify gross structural features (such as headings) and syntactic units (such as dependent clauses or sentences). SGML describes a markup language independently of what the markup is intended to do. Thus, to understand and act upon the markup, additional semantic information is needed.

The general idea behind markup languages is based upon three aspects:

1. The primitive level: text is composed simply of streams of symbols or entities.
2. A higher level: where text is composed of representations of objects of various kinds, linguistically or functionally defined. Each object appears in specifiable relationship to other objects, they may be included within each other linked to each other by reference or simply presented sequentially. These are also known as elements.
3. The grammar: which defines how elements may legally be combined (Document Type).

SGML addresses, the specification of markup allowance, the markup requirements, and how to differentiate markup from text. The instructions (in SGML) needed to process a document for

some particular purpose are sharply distinguished from the descriptive markup, which occurs within the document.

Furthermore, in a SGML markup document the logical parts are clearly marked, thus creating a contrast to other formats (such as RTF) where a document can be read, understood, and edited with little effort (van Herwijnen, 1994). But, it is the markup that allows the portability of the document and the ability to be processed by different applications.

Some facts that make SGML a powerful tool are:

Hardware independence, thus easy interchange of text
Separation of structure and layout
Software independence, giving an open-endedness to data
Unambiguous format, enabling database storage and retrieval
(van Herwijnen, 1994)

It is worth noticing that SGML is not vendor specific. Its descriptive markup can be processed by many different pieces of software, each of which can apply different processing instructions to those parts of the markup, which are considered relevant. The instructions needed to process a document for some particular purpose are sharply distinguished from the descriptive markup that occurs within the document. Also, SGML is an open standard controlled under ISO rules, which generate and modify standards in a process that is open to the public through participation in national standard bodies (van Herwijnen, 1994).

There are some intrinsic components to an SGML document. These include a formal specification of the markup constructs used, expressed in SGML, but it can also include a non-SGML definition of semantics, application conventions, and/or processing (Goldfarb, 1990). Nonetheless, there are three formal parts to an SGML Document, it consists of an SGML prolog and a document instance. The prolog contains an SGML declaration and document type definition. The document type definition contains the elements and entities declaration (Practical SGML, TEI, 2000).

1. *The SGML declaration* specifies basic facts about the dialect of SGML being used such as the character set, codes, length of delimiters, etc. The SGML declaration is usually invisible to the user.
    - Is usually common to all documents in an SGML installation
    - May or may not be part of the document
    - Gives precise details on how SGML was applied to that document.
    - Defines which character set is used

2. *The Document Type Definition* specifies the document type definition against which the document instance is to be validated. It is also invisible to the user and it only requires the name of the document type to be specified before the document is to be validated. It defines the rules for marking up a class of documents.
    - Defines the structure of a document

- Is different for each set of documents with a different structure
- Is written in SGML
- May be stored externally to the document

3. *Document Instance:* Is the content of the document itself. It only contains text, markup, and general entity references.
    - Contains data
    - Contains markup
    - Includes reference to the Document Type Definition

SGML only describes SGML data, the DTD and what is the role of the SGML parser. These parts are usually hidden inside the software that the person interacts with as a user (van Herwijnen, 1994).

A parser is a piece of software which can take a document type definition and generate from it a software system capable of validating any document invoking that document type definition. Is a special purpose program that can be used to process a document claiming to be of a particular type and check that all the elements required for that particular type are indeed present and correctly ordered. With this, different documents of the same type can be processes in a uniform way. Programs can be written to take advantage of the knowledge encapsulated in the document structure information, and can thus behave in a more intelligent fashion.

6.1.1 Document Type Definitions – DTD-

A DTD is a set of markup declarations that apply to all documents of a particular type. It formally defines: what are the legal constructions of a given markup language, what are the rules, what data are accepted, how and in what order. The designer of a DTD generally has to trade off generality of use with accuracy of error detention.

The three most important kinds of declarations that can occur in a DTD are:

1. An element declaration: which defines the elements that can occur within each element, and in what order.
2. An attribute definition list declaration: defines the attributes that can be specified for an element, and their possible values.
3. An entity declaration: defines the entities that can be referred to in documents of this type (Goldfarb, 1990).

Rules are the first stage in the creation of a formal specification for the structure of an SGML document, or document type definition. Every document type definition is an interpretation of a text. At present SGML is used in environments where uniformity of document structure is a major concern and a need. In the production of technical documentation, it is of great importance that sections and subsections should be properly nested, that cross-references should be properly resolved and so forth.

So, by making rules explicit, the person in charge of encoding reduces his or her own burdens in making up and verifying the electronic text, while also being forced to make explicit an interpretation of the structure and the significant particularities of the text being encoded.

A DTD is expressed in SGML as a set of declarative statements using syntax defined in the standards. It specifies:

a. The generic identifiers (GIs) of elements that are permissible in a document of this type
b. For each element, the possible attributes, their range of values, and defaults
c. For each element, the structure of its context, including:
    1. which sub-elements can occur and in what order
    2. whether text characters can occur
    3. whether character data can occur

However, a DTD does not specifies:

1. The delimiters that are used to indicate markup
2. Anything about the ways in which the document can be formatted or processed.

The three parts that are described in the document type definition are:

6.1.1.1 Elements

Element is the SGML standard for a textual unit view as a structural component. It is why everything in an SGML document is delimited explicitly in some way, why a document is decomposed into elements of a named type, and why a kind of textual grammar can be defined.

SGML provides no way of expressing the meaning of a particular type of element, other that its relationship to other element types.

There are three important aspects of textual objects that need to be recognized:

1. Their extent: points in the text where elements begin and end
2. Type: the category to which elements are assigned
3. Context: their relationship to other elements within the document

All the elements in a given document type may be arranged into a hierarchic structure, arranged like a family tree with a single parent at the top and many children at the bottom. SGML does not cater for the case of more or less freely floating elements that can appear at almost any hierarchic level in the structure, and it does not cater for the case where different elements overlap or several different trees may be identified in the same document. However, to be able to navigate between elements, one has to know the relation between elements. One can say about an element: (1) which are the instances of it (the elements that can occur within other elements), and (2) if it may be decomposed into other elements. In other words, you can specify who is the parent and who are the children elements.

Another characteristic of elements is the GI, or general identifier. It is the technical term for the name of an element type.

As well as other components of SGML markup text, elements are delimited by tags. The primary function of the start and end tag within a marked up text is to indicate the extent of a particular object or textual component within it. In addition, the tags identify the category or type of the element, which they delimit, by supplying a name for it. This way, an SGML-aware processor can easily identify the start and end of all elements.

The content of a document element of a particular type may consist simply of running text, perhaps including entity references, or it will contain other embedded document elements. Occasionally it may have no content at all. This means that an element may be empty, or it may contain simple text. However, elements of one type will be usually embedded within elements of a different type.

6.1.1.2 Attributes:

Are used to describe information, which is in some sense descriptive to a specific element occurrence but not regarded as part of its context. Attributes are element's properties other that it's content. They are defined by an attribute definition list declaration that associates an "attribute definition list" with the element type (Goldfarb, 1990).

Attributes are declared in SGML in the document type declaration. If an element is regarded, or has been defined as having attributes, the attribute values are supplied in the document instance as attribute-value pairs inside the start tag for the element occurrence.

There are categories of information associated with a particular type of element, but not contained within it. Attributes are associated with particular element occurrences by including their name and value within the start tag for the element concerned (van Herwijnen, 1994).

The purpose of using attributes are: 1) they enable an identifying number or name to be associated with a particular element occurrence within a text; and 2) they enable additional information violating its integrity (van Herwijnen, 1994).

However, the most common use for attributes is to identify element occurrences. And most important all attributes must be defined in advance

6.1.1.3 Entities:

Within a system, a single document can be stored in several parts, each in a separate unit of storage, called an entity. An entity could be a file, a data set, a variable, a data stream object. Thus, it is a named segment of text considered entirely independent of any structural or categorical classification that it might have (Burnard, 1996). That is why it has resemblance to a programming language variable (Goldfarb, 1990).

Entity is a named part of a marked up document irrespective of any structural considerations. This is a way of SGML to provide a simple and flexible method of encoding and naming arbitrary parts of the actual content of a document in a portable way.

The definition of an entity associates a name with a particular string of bytes, which may be the representation of some characters in a particular computer encoding or held in a system-defined container of some kind. Within an SGML document, entities are represented by reference, using the defined name. Once an entity has been declared, it may be referenced anywhere within a document.

To provide character standardization, the SGML solution is to encode characters not available in the particular character set used for document transition by means of entity reference. The advantage of entity reference solution is simply that it forms part of a single and consistent convention, comprehensible without resort to special purpose documentation (Burnard, 1996).

Separate entities are connected by entity's references that can occur in the document's markup. An entity reference is a request for text, (the entity) to be imbedded in the document at the point of the reference. The entity could have been defined either earlier within the document or externally.

The convention used to differentiate an entity in markup text is as follows: entities are delimited by an entity reference open (or a parameter entity reference open), an ampersand "&" or a percent sign "%"; and a reference close –a semicolon ";".

To finalize is worth mentioning that the purpose of the SGML markup is simply to indicate that a cross-reference exists; it does not determine what the processor is to do with it. Text-oriented database management systems typically use inverted files indexes to point into documents, or subdivisions of them. A search can be made for an occurrence of some word or word pattern within a document or within a subdivision of one. Meanwhile subdivision of input documents will, of course, be closely related to the subdivisions specified using descriptive markup. Thus it is simple for textual database systems to take advantage of SGML-tagged document. Research work is currently going into ways of extending the capabilities of existing (NON-TEXT) database systems to take advantage of the structuring information made explicit by SGML markup (TEI, 2000).

6.2 eXtensible Markup Language –XML-

Extensible Markup Language is a simplified metalanguage, derived from the Standard Generalized Markup Language (SGML). It emerged as the standard for self-describing data interchange in Internet applications. XML was developed by the World Wide Web Consortium (W3C) in 1997, and many major platform vendors (IBM, Microsoft, Netscape, and Sun Microsystems) are implementing this new technology (Glushko, 1999).

This markup language provides a way to extend data formatting and delivery for specific uses. Software agents and search engines have difficulty using information because it is not semantically encoded. With XML information and services can be encoded using a meaningful structure and semantics that computers can readily understand. Using XML any agent with the proper authorization will be able to obtain computer-interpretable data sheets, price lists, and inventory reports through the web or email (Glushko, 1999). Goldfarb says that the goal of XML is the digital representation of document. To digitally represent a document, is to compose the documents in some kind of computer-readable notation so that a computer can help us store, process, search, transmit, display and print them (Godfarb, 2000).

A markup language is not a language in the same sense C, C++, COBOL, of FORTRAN are languages. Instead, is a set of rules that define how text or documents should be "marked up", is a set of rules that define HOW we add meaning to areas of a document (Homer, 1999).

XML was developed because of the needs created by the Standard Generalized Markup Language (SGML) and the HyperText Markup Language (HTML). The three of them are markup languages, but SGML and HTML are mostly related to page layout formatting. SGML was designed to separate information from formatting, while providing an open standard for creating and storing documents. In addition, SGML is both language and a set of rules for creating specific data applications. SGML requires a language interpreter with a high level of complexity, therefore it is not supported by Web or Web Browsers.

On the other hand, HTML was developed as a page layout formatting language. Tags are predefined, and the language is not strict in the sense that allows programming short cuts to be taken. HTML was created specifically for displaying documents on the Web using web browsers. And even though it is a standard, the different web browser developers have enable, or disabled, some of the features of HTML.

XML is "extensible" because it allows programmers to define tags that best represent the data to be transmitted, and the actions that are going to take place. XML's power derives from its extensibility and ubiquity: anyone can invent their own tags for particular subject areas, defining what they mean in document type definitions (DTD), and in Schemas. Content-oriented tagging enables a computer to understand the meaning of data. (Glushko, 1999). XML is designed to describe the structure of text, not how it should be displayed, thus why its does not have a fixed set of tags.

Because of this extensibility, XML data can be customized by application. The form it takes at the display point depends on how it is processed (Computer Design, 1998). However, since

programmers create their own tags, a browser, or any other application, may not interpret the XML correctly. For this reason, the W3C has defined the XSL, or extensible styling language. XSL is an XML based language that allows transformation of XML data to any other type of text output (Homer, 1999).

XML provides a way to extend data formatting and delivery for specific uses. It will become a delivery standard for data with high degrees of reuse and automation, because users are able to separate the context of the data from the application (Computer Design, 1998). This maintains accordance with object-oriented programming and reusability of object in different applications. More and more, companies are reusing, recycling and remanufacturing products, materials and resources, and computer resources can not be left out.

According to Ian Agravan, president of Agravan Systems XML will probably be the primary method of getting data from embedded devices to applications at the other end of the connection (Computer Design, 1998). XML eliminates the need for custom interfaces with every customer and supplier. Today, much of the Business to Business (B2B) information is exchanged through EDI (Electronic Data Interchange) messages. Traditional EDI is complex and expensive, because most messages travel over proprietary networks. Moreover, EDI's brittle syntax necessitates a custom integration solution between each pair of trading partners. For this reason, there will be an increasing use of XML/EDI message format. Such messages will be more economical that traditional EDI messages, while being easier to validate and translate into the formats needed by applications at each end of the exchange (Glushko, 1999).

XML is for the digital representation of documents. It puts documents in some kind of computer-readable notation so that a computer can help us store, process, search, transmit, display and print them (Goldfarb, 2000). XML is designed to describe the structure of text, not how it should be displayed (Homer, 1999).

An XML Text refers to the combination of character data and markup, not only character data. XML text is comprised of elements, which are enclosed between an initial tag <tag>, and an end-tag </tag>. An element has three parts: a start tag, an end tag, and the text it encloses. The markup, or tags, identifies each component (element) of the document in ways that the computer can understand. Each element is an instance of an element type (see figure 6.1) and each element can contain *attributes*. Attributes, as in HTML, define the properties of elements (Goldfarb, 2000; Homer, 1999).

XML's structure is what it is called a *tree structure*. Elements represent a logical component of a document, but elements can contain other elements, or can contain character data. In a tree structure the root element (or root tag) is the outer tag, is the element that contains all the others. In our example the root element (or root tag) is <Inventory>.

One has to be careful when defining elements. Each element describes the logical role of it, of the abstraction it represents. This abstraction has to be descriptive enough and suitably chosen so that particular uses of the document can be completely automated using computer processes that act on the elements (Goldfarb, 2000).

```
<Inventory>
        <product>
                <description>
                        <id> 5463965 </id>
                        <name> exhaust valve </name>
                </description>
                <quantities>
                        <initial  measure = "units"> 2,000 </initial>
                        <requested measure = "units"> </requested>
                        < final measure = "units"></final >
                </quantities>
        </product>
</Inventory>
```

Figure 6.1: XML code example

In XML there is a special processing instruction tag: <?xml>. This tag should be used at the first line of each XML document, and it can be used to identify version and language. For example:

<?xml version="1.0" encoding= "iso-8859-1" ?>

XML is still in version 1.0 and the encoding refers to the character set used in the data file, and in the XML document (Homer, 1999). XML character set is Unicode, it allows authors to use standard plaint text editors to create XML. Unicode, is a character set where the first 128 characters are compatible with ASCII. The Unicode character encoding (UTF 8) first 128 characters, are the same as the 7 bit ASCII characters (Goldfarb, 2000).

Another important piece of XML, are *entities*. While XML elements describe the document's logical structure, entities keep track of the location of bytes that make up an XML document, of the physical structure of the document. Entities have name and references, and the processor replaces the entity reference with the entity itself. However, an entity is not a file. Entities help to break up large files to make them editable, searchable, downloadable and otherwise usable on the ordinary computer system (Goldfarb, 2000).

Hypertext is a form of entity, this type of text could be reuse in many different contexts automatically. But one of the important things is that an update in one place would propagate across all users of the text. This type of text reuse is referred to as an external entity (Goldfarb, 2000).

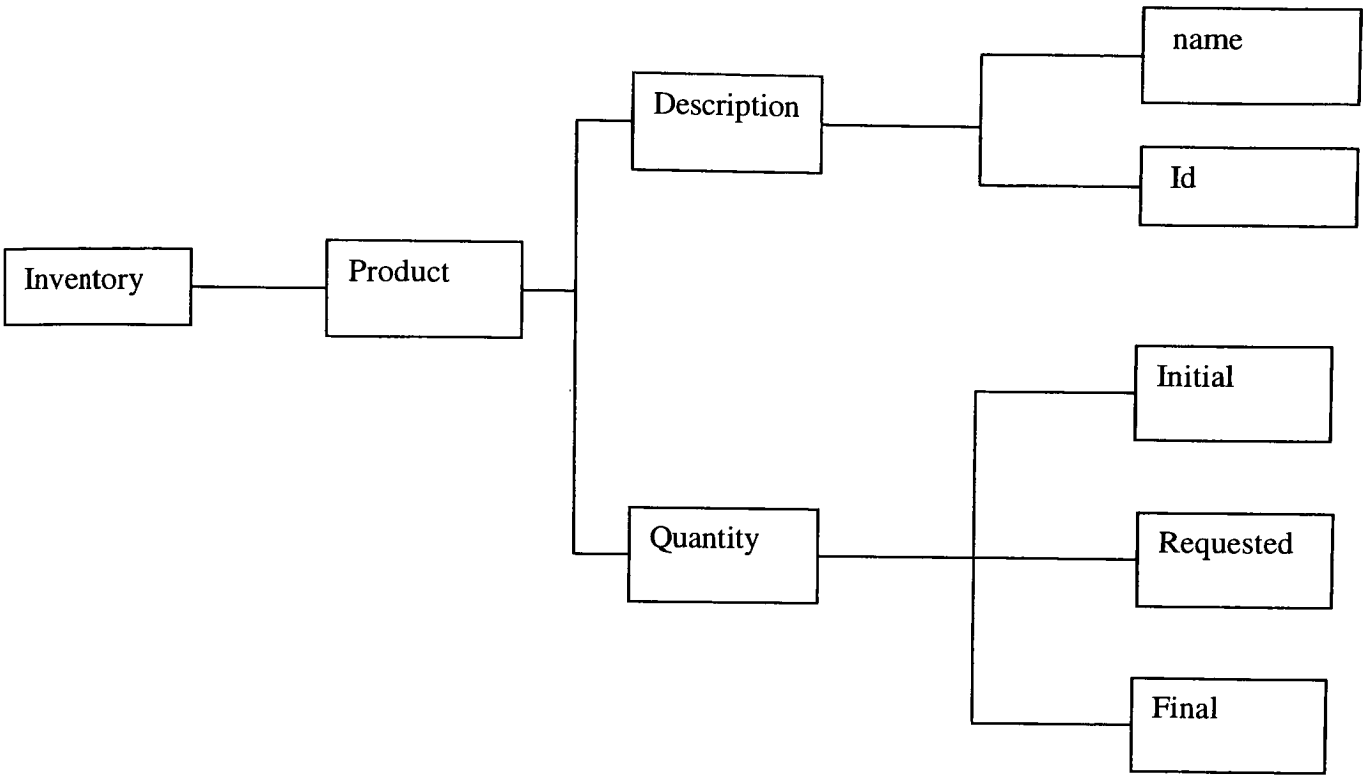


Figure 6.2: XML's tree structure

6.2.1 Document Type Definition in XML

In the effort to maintain XML as a standard, every detail has been defined. Document Type Definition (DTD) is a series of definitions for element types, attributes, entities and notations. It declares which of these are legal within the document and in what places they are legal. It is a text file that defines the structure of a XML. A DTD is the formal specification, or grammar, for documents of a given type, describing the elements their attributes, and the order in which they have to appear (Goldfarb, 2000; Homer, 1999; Computer Design, 1999).

The validity of an XML document depends on the DTD. According to Goldfarb, a document is valid if it declares conformance to a DTD in a document type declaration, and actually conforms to the DTD (Goldfarb, 2000).

An equivalent for DTD has been proposed by Microsoft (Homer, 1999), called Schemas. As with DTD, schemas specify which elements are allowed in a document and can turn a well-formed XML document into a valid XML document. A valid XML documents declares conformance to a DTD, or schema, in a document type declaration, and actually conforms to the DTD or schema (Goldfarb, 2000; Homer, 1999).

However, Goldfarb, (Goldfarb, 2000) gives a word of caution. He says " A DTD is a concept, markup declarations are the means of expressing it. The distinction is important because other means of expressing DTDs are being proposed... "

The DTD for the Inventory XML would describe the quantity of elements from the INVENTORY document (INVENTORY+). Each inventory element is made up from five other elements (product_description, prod_id, etc.), and each of these sub-elements contains character data (CDATA) (Homer, 1999),

```
<!ELEMENT DOCUMENT (INVENTORY+)>
<!ELEMENT INVENTORY (product_description, prod_id, prod_name, initial_quantity,
                     requested_quantity)>
<!ELEMENT product_description (CDATA)>
<!ELEMENT prod_id (CDATA)>
<!ELEMENT prod_name (CDATA)>
<!ELEMENT initial_quantity (CDATA)>
<!ELEMENT requested_quantity (CDATA)>
```

Figure 6.3: DTD for Inventory XML

Because you can define your own tags, a problem one could encounter when writing XML is that there could be more than one tag with the same name. To avoid confusion between named tags, there is defined a *Namespace*. Namespaces uniquely identify the schema to which elements belong. Using a XML namespace attribute (XMLns), and the Uniform Resource Identifier (URI), you can differentiate two elements with the same name. Namespaces allow ensuring that the data is interpreted correctly and it allows the XML to be distinct from any other XML that may content the same elements or attributes (Homer, 1999).

6.2.2 Application Processing Interfaces for XML

Document Object Model, DOM is an Application Processing Interface defined for both HTML, and XML. It defines the logical structure of documents and the way they can be accessed. It is used to read, write, and transmit XML on a web server. In other words, it is a standard way in which we can access and manipulate the XML structure (Goldfarb, 2000; Homer, 1999).

Another API for XML is the Simple AP for XML, or SAX. It is an event-based parser that allows getting small parts of the data as they come in. It lets the application work on the bit of the data that the parser finds at each "event" in the document (Goldfarb, 2000).

6.2.3 EXtensible Style Language (XSL):

Style sheets allow us to attach visual styles to documents without altering the markup language. XSL is an XML based language that let you transform XML data to a specific display. One can convert XML to HTML, or from XML to any type of text output. Moreover, whenever you want to display XML data you need to style it in some way. The way XML is transformed is by matching elements and outputting text and element values (Goldfarb, 2000; Homer, 1999).

When working with XSL, you also work with templates. These are a set of rules that match elements or attributes in the XML. Within a template you can loop through elements and attributes, apply other templates and perform other types of processing (Homer, 1999). However, because the style is describe in the stylesheet, applications that are not interested in style can ignore the XSL (Goldfarb, 2000).

6.2.4 XML Applications:

As stated before, the goal of XML is to represent in a digital form any kind of document. However, another important characteristic of XML is that the same software can process a diversity of documents (Goldfarb, 2000). Two of XML applications are Presentation Oriented Publishing and Message-Oriented Middleware.

By using Presentation Oriented Publishing (POP) and XML, one can capture what is IN a document file, not how it is supposed to look. Then, using a stylesheet, the POP users give the instructions as how the document should be formatted. And since XML is a digital representation of the document, you can display the information using several different "looks" by having different stylesheets (Goldfarb, 2000).

Message-Oriented Middleware (MOM) are documents generated by programs for other programs to read, but mostly it processes data. Goldfarb, (Goldfarb, 2000) states that the importance of XML MOM is that introduces a middle-tier into the traditional client/server interaction, and it is this middle tier the one that acts as a data integrator. With this middle-tier the client can read data from anywhere, but only has to understand data that is in XML documents, in contrast with having specialized programs to access particular databases.

6.3 XML Three-Tier Architecture:

A three-tier architecture consists of a two tier networking model (client/server), plus a single middle-tier server that can be an intermediary that aggregates data from multiple sources and presents all of it at once to the client (Goldfarb, 2000).

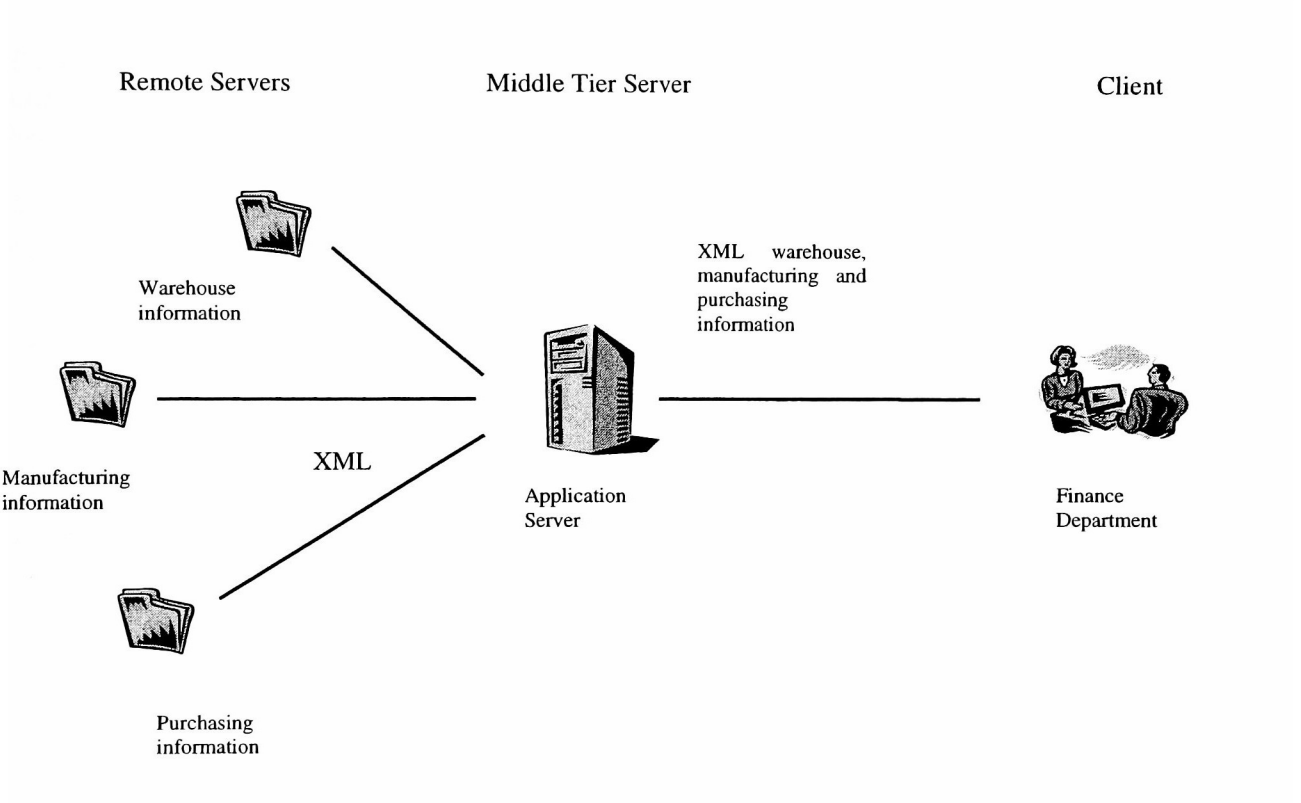


Figure 6.4: Three-Tier Architecture

In an XML three-tier architecture the middle tier acts as a data integrator. It does all the talking to the data sources and sends their messages in XML to the clients. Thus, the client can read data from anywhere, but only has to understand data that is in XML documents. XML provides the *metadata* that was in the original data source schema (Goldfarb, 2000).

For many applications, the middle-tier server collects all of the relevant data at once and sends out a single document to the client. Querying, sorting and other processing then take place on the client system, allowing overhead and traffic to be cut down, and improves performance and satisfaction of the end-user (Goldfarb, 2000). Therefore, the role of the middle tier is to gather information from data sources and deliver it in a consistent manner to clients.

Since XML is a "young" technology, there are some parts adjacent to XML that are not yet part of the standard (Homer, 1999):

Extensible Hypertext Markup Language (XHTML): It applies the strictness of XML to HTML. For example, it requires open and closing tags for every element, and enforces the idea that attributes have to be enclosed in quotes.

Xlink and Xpointer: They allow linking between XML documents. This extends the functionality of existing linking schemes, providing a flexible and efficient solution.

Extensible StyleSheet Lanugage (XSL): XSL is still not a standard. One of the elements of XSL that is being designed is the formatting of objects. These will introduce a set of tags for formatting objects, providing a display-independent description of the formatting.

BizTalk: This framework is currently being designed by Microsoft. The idea behind it is to define a standard set of XML tags for common business scenarios. (www.biztalk.com)

XML is a self-describing language. Applications are very flexible in their ability to receive, parse and process diverse information. It gives independence to the client and server sides, because it provides a way of structuring the information, but it does not regulates how the information has to be displayed or utilized. The fact that users can define their own tags gives some sort of uniqueness to their XML documents. Users can customize their XML files according to the applications being used, resulting in a more efficient communication network.

7. IDL – XML Architecture for Enterprise Integration

Before the concept of a paper-less enterprise became known or applied, information was transmitted from one department to another by means of paper reports. As a consequence, reprocessing of information was necessary, and resulted in human error as a product of the need to re-enter data. This is not to state that the task is better performed by machines or computers, but that humans cannot always maintain focus on small details for extended periods of time. However, if information is shared through a computer network, data has to be fed into the system only once, reducing or eliminating the potential for erroneous input.

Moreover, some of the issues that integration confronts today are the incompatibilities between systems, platforms, hardware, software, and the physical distance between clients or applications and servers. For example, raw data from a data acquisition system cannot create an Excel table. There is a need for software to convert the raw data into a usable form understood by the application. It is difficult for a Windows application to interact with a robotic workstation without additional program interfaces. Similarly, it is impossible to have Unix and Windows interact without the assistance of Samba, CORBA or another type of middleware.

When dealing with manufacturing environments, one finds different types of hardware and software trying to interact with each other. Such objects include sensors, controllers, robots, planning systems, CNC, design software, and data acquisition. Each of these elements has their own characteristics and their own tasks to perform. Some communicate with each other, but not without proper protocols and software. When these elements need to be integrated with other types of software, Material Requirement Planning (MRP), Manufacturing Resource Planning (MRP II), and any other planning, simulation, design, manufacturing, or finance software, a great programming and monetary effort is necessary. Client/Server technology and Electronic Data Interchange (EDI) have been utilized in order to accomplish some integration within these areas. But the main problem faced today is that most integration efforts have led to the formation of Islands of Automation. This results in the creation of different databases, one independent of the other, and sometimes with the same information being stored. By having several databases, multiple updates have to be performed, rather than a single update when information needs to be modified. This creates conflicts of versions, due to the inability to accurately perform multiple updates and it might be due to the unavailability of a specific database at a particular time.

The Society of Manufacturing Engineers says that data communication improves the organizational and personnel efficiency in a corporation. As in any team, in systems integration each part has to be able to transmit its knowledge, and to receive and understand the knowledge of the other members of the group. The information flow, the structure of the information in an enterprise must have three qualities: performance, concurrency and comprehension. Also, it has to be standardized across the enterprise in order to allow manufacturing applications to interchange information in an efficient manner, and to give users access to familiar applications when ever possible.

The architecture proposed in this work provides a methodology of data storage, data retrieval, and data processing in order to provide integration at the enterprise level. There are four layers of interaction in the proposed IXA architecture. The name IXA (IDL – XML Architecture for Enterprise Integration) is derived from the standards and technologies used to define the layers and corresponding functions of each layer (See figure 7.1).

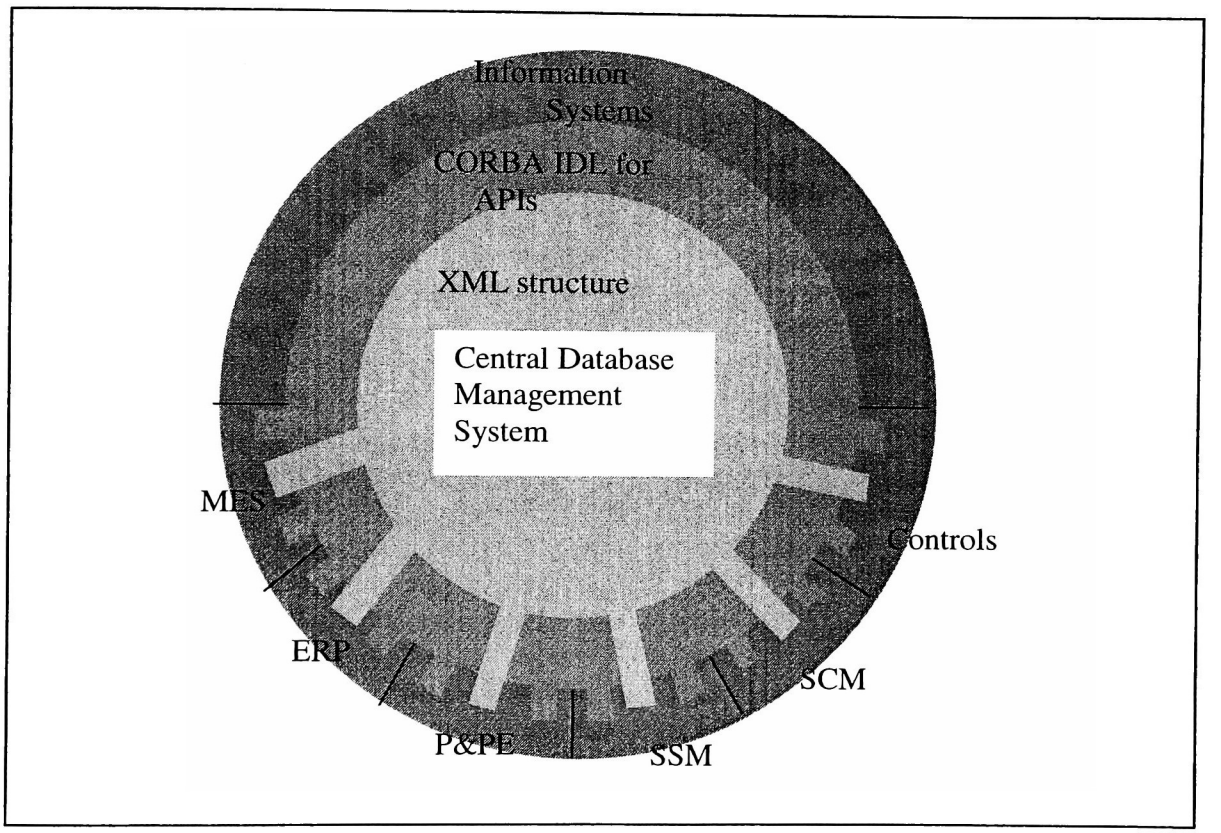


Figure 7.1: The four layers of IDL-XML Architecture for Enterprise Integration

The first (outer) layer represents the different Information Systems (IS) of an enterprise, i.e. Manufacturing Execution Systems (MES), Enterprise Resource Planning (ERP), Supply Chain Management (SCM), Sales and Service Management (SSM), Product and Process Engineering (P&PE), and Controls. These are systems that collect, store, retrieve, and utilize the information throughout the enterprise. They vary in location, type of data managed, language, process, and platform.

Earlier we defined and explained the use of each of the information systems. Moreover, we defined the CIM architecture and mapped the different levels of the CIM model to each of the information systems modules defined by the MESA context model. Each layer of the CIM

architecture needs to communicate with at least two more levels: the level below and the level above it. Because of the difference of data manipulated by each level, the decision time frame, and the interaction with other equipment (i.e. the floor level interacts with actuators and sensors) the data are diverse and thus difficult to share. The information shared by applications has to be encoded in such a way that both receiver and sender could make use of the data. In other words, information has to be encoded in a "standard" way.

However, each information system could integrate different applications and still be independent and isolated from the other information systems. Since an enterprise needs some functionality from each of the six information systems previously defined, the standardization of information and communication must take place through the entire enterprise. Bhatt, quotes Hammer and Champy in the following: "By linking inter-organizational, inter-functional, and inter-personal levels of the processes through information systems, business are not only automating their activities, they are also reshaping and improving their business processes" (Bhatt, 2000). Therefore, there is no doubt about the importance of the interaction between ERP, MES, and Controls, as well as with the other three systems. But at the same time, seamless interaction between them is unrefutable. Hence we propose the CORBA interface layer into the IXA architecture.

The second layer, the CORBA layer, provides an Application Processing Interface (API) for those queries that expect a process to be performed in order to obtain information. It provides the ORB to interact between the different information systems, and also provides the means to store information that would not add any long-term value to the enterprise. Because of the variety of controllers that interact in a manufacturing floor and the diverse languages used to deliver data, processing information becomes complicated. Applying the CORBA standards eliminates the difference of languages between applications. Therefore, language differences would not affect processing between applications, neither platform not physical distance. CORBA's IDL and ORBs would perform the communications between the information systems. The Object Request Broker would make requests to - and receive responses from- the different applications within an information system.

As explained in the CORBA section, the ORB would intercept a request from the client to an application, it would find a server application that could process the request, pass it the parameters, invoke its methods, and return the results to the client. This way, information systems can share data that does not need to be store, and can call upon applications and processes. With the use of ORBs, several requests can be handled at the same time. The CORBA architecture calls for several ORBs that communicate between applications and between each other.

Another important piece of the CORBA layer is the Interface Definition Language. The IDL provides the parameters for the communication between ORBs and applications. In other words, it defines the interface of an application, what functions are supported by it, the output and any other characteristics of importance.

IDL interfaces will be developed for all enterprise programs to allow ORBs to interact between:

- servers and clients
- different information systems
- Enterprise and customers, and
- Enterprise and suppliers.

By the use of CORBA in the IXA architecture, the means for communicating among the different information systems, and even between the enterprise and the customers, are established.

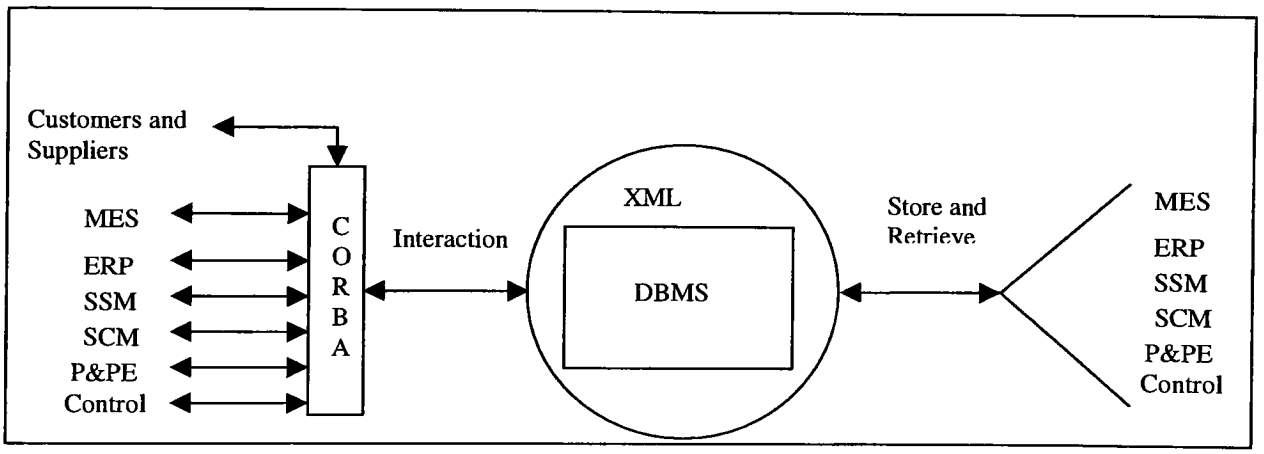


Figure 7.2: Interaction of the different layers

Figure 7.2 depicts the interaction between the different layers. The CORBA layer regulates the interaction between the IS, the outside world and the database manager. The interface that is described with the IDL is necessary in order to provide language independence between the pieces of the Enterprise System. On the other hand, and as it is explained in the following paragraphs, the XML is used to digitally encode the information to make it available to the enterprise. Access to the Database is not only restricted to either CORBA or XML, but to both. The way the IXA architecture is defined allows the systems to go either through CORBA and then XML, only through CORBA, or only to XML. The path that the IS takes, depends on the requirements of the application, the needs for data storage and retrieval, and the operations to be performed on the data.

The different situations or paths for the information interchange are presented in figure 7.3. The Information Systems (IS) can communicate with the database via CORBA and XML, or using only XML if its to store or retrieve data. As well, they can interact with other IS or with the outside world using CORBA interfaces and brokers.

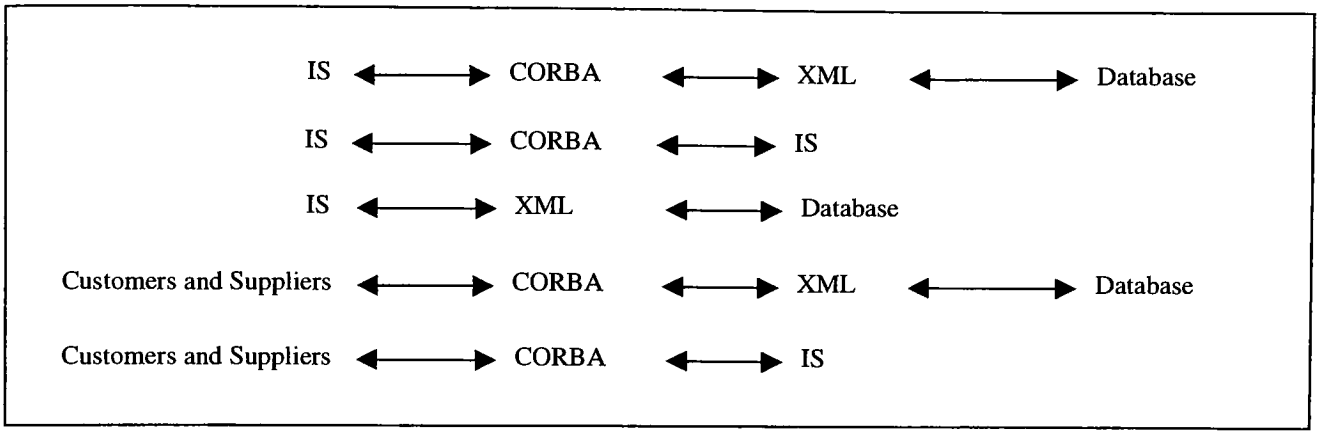


Figure 7.3: Different information sharing situations with the IXA model

The XML layer (the third layer) gives the structure for the information that needs to be stored for further analysis or for later access. Because the information stored in a database has to be accessible to every one in the enterprise, it must be encoded in such a way that facilitates the use of it by any program and any client. The CORBA layer of the architecture defines the way applications will interact with each other, but it is the XML layer the one that provides the structure to store and retrieve information.

XML does not define how data should be presented, but how it is structured. It provides a robust enterprise wide structure to organize the data, based on XML's tree-structure or hierarchy. The XML will provide a robust enterprise-wide structure to organize and store the data. In the tree structure elements contain other elements, which could contain other elements or data. This structure allows information to nest on many levels, enabling a structured search. Navigation through a structured tree is simplified compared to the navigation through a relational database table. A tree structure relates the different levels as parent and child, thus allowing a true relation between the different elements under an XML structure.

However, even though tree-structure facilitates navigation through the information, the tags that make up a specific XML for an enterprise must be standardized. One of the advantages of XML over other markup languages is the ability given to designers to customize tags and attributes. Nevertheless users or related applications need to understand the customized tags before using them (Shian and Wofl, 2000). By providing an enterprise-wide set of tags (appropriate for each

different enterprise or business) one can structure the way data will be accessed. The different information systems will retrieve the data using the appropriate XML tools (i.e. XSL), thus providing the means to represent the information digitally for each part of the enterprise in order to make use of it. The standardization of tags for each enterprise is a must. Each enterprise must define the tags, attributes, elements, and entities to be utilized to store and retrieve the information.

XML provides a way to encode the information, to digitally represent the documents. It provides a medium to move data across dissimilar systems with quick transformations of the data; it provides a semantic structure to the data. Overall, having XML tags to encode the information before storing it, facilitates the declaration of the data, and provides a more meaningful search of the stored information.

The last component of the IXA Architecture is the Central Repository. Databases represent an important infrastructure for computer applications, it is the location where data is stored, and later retrieve when making sales, doing a search, or processing information. The centralized information is stored here. It will have its own Database Management System (DBMS), which will enable and regulate the access to the database by the different information systems. The central repository eliminates data redundancy and adds a tighter control on data access.

Since the late 1960's and early 1970's relational databases have been the main stream in database management systems. However, the architecture proposed here, defines the central repository as having a tree structure (or hierarchy) to store the data instead of the table that is characteristic of the relational databases. Relational databases can still exist within this framework at a lower level.

A central repository is nothing more than a common database, with its own Database Administrator System, which is a program that enables, and regulates the access to the database. When data is stored in this central repository, systems would have access to the data depending on the needs of the system or applications. Because of the privileges of systems the database manages, some systems can not alter the data, some can not retrieve the data, and others would have full control of the data. The importance of having a central repository where all information is stored comes because of the need to have access of all information through the entire enterprise. A central database could be seen as a hub for key data, assuring that each process maintains its independence, yet be accessible to other applications or users (Bourdeau, 1991).

Consequently, the proposed **I**DL **X**ML **A**rchitecture for Enterprise Integration (**IXA** for Enterprise Integration) utilizes the standards and technology to provide the base for information sharing where language and platform differences act as a barrier. This model provides the means for communication between information systems via CORBA. It structures the data using XML hierarchy to allow a better navigation through digitally represented documents and information, and provides an API interface (IDL) to process information between client and server.

The IXA model supports the integration not only of manufacturing systems, but also of other systems with in an enterprise. It addresses the main concerns of information sharing: Performance, since the data would be delivered in a single communication. Concurrency, the Information would be access and deliver according to the needs of applications to prevent the locking of information and to allow many applications to work at the same time with the database. And Comprehension, because both receiver and sender would understand the data. It also, provides an organized structure for information sharing among the diverse components of an enterprise, and achieves efficiency and overall improvement of the processes.

8. References

"Controls Definition & MES to Controls Data Flow Possibilities" *MESA International White Paper #3* August 1995 <www.mesa.org>.

"CORBA-based Machine Control White Paper". *Object Management Group (OMG)* <www.omg.com> (Jan 11, 2000).

"Directions in CIM for Semiconductor Wafer Fabrication". Solid State Technology February 1994: 29-37.

"Enterprise Resource Planning (ERP): How to Understand and Use the New Capabilities". *Professional Development.* < http://profdev.sjsu.edu/> (Jan 19 2000).

"Enterprise Resource Planning Integration". *Raytheon Automated Systems-ERP Integration* November 10, 1999 <http://www.esys.com/rec/autosys/sub_erp.htm> (Jan 19, 2000).

"Enterprise Resource Planning Overview (Course Description)" *Joint Information Systems Committee (JISC)* April 21, 1999. <http://www.jisc.ac.uk/info_strat/events/erp.html> (Jan 19, 2000).

"ERP Glossary of Term". *Intermec Technologies Corporation.* <http://www.intermec.com/erp/erp_glos.htm> (Jan 11, 2000).

"ERP Systems-Evolution" *Query Resutl to Ask Jeeves.Com* <www.askjeeves.com>.

"ERP's Second Wave: Maximizing the Value of ERP-Enabled Processes". Deloitte Consulting

"John Parsons: The Man Behind Numerical Control". Manufacturing Engineering Jan 1982: 147+

"MES Explained: A high Level Vision". *MESA International White Paper #6* September 1997 <www.mesa.org>.

"MES Funcitonalities & MRP to MES Data Flow Possibilities". *MESA International WhitePaper #2* March 1997 <www.mesa.org>.

"NIIIP SMART/MESA Response to Manufacturing Domain Task Force RFI-3 Manufacturing Execution Systems". *Object Management Group (OMG)* May 5, 1998 <www.omg.com> (Jan 11, 2000).

"On Track® Manufacturing Execution Systems". *Real World* Technology <http://www.ontrack-mes.com> (Jan 19 2000).

"TEI Guidelines for Electronic Text Encoding and Interchange". *Electronic Text Center at University of Virginia,* < http://etext.virginia.edu/bin/tei-tocs?div=DIV1&id=SG> (May 1, 2000).

"What is ERP". *FMM Dancom Systems.* <http://www.fmm.org.my/fmm-ds/erp.htm > (Jan 19, 2000).

"White Paper ERP for Manufacturing". *Enterprise Resource Planning for Manufacturing.* <Cambashi limited>.

Adasoft Group <www.adasoftgroup.com>

Adelsberger, Heimo, et al. Information Management in Computer Integrated Manufacturing. New York: Springer, 1995

Allegri, T.H. Advanced Manufacturing Technology. Tab Books: 1989.

Attaran, Mohsen. " Information Systems Play a Critical Role in CIM Success". IIE Solutions Dec 1995.

Bhatt, Ganesh. "Enterprise Information Systems Integration and Business Process Improvement Initiative: An Empirical Study". IIE Solutions Conference. Cleveland, June 2000.

Ben-Shaul, Israel, and Gish, James, and Robinson, William. "An Integrated Network Component Architecture". IEEE Software September/October 1998.

Blair, Raymond N. Elements of Industrial Systems Engineering. New Jersey: Prentice Hall, 1971.

Bourdeau, Richard. "The Relational Database: Backbone of CIM solutions". I&CS. September 1991.

Bos, Bert. "XML in Ten Points" <www.w3.org/xml/1999/xml-in-10-points> (March 09 2000)

Bosak, Jon. " Media-Independent Publishing: Four Myths about XML". Computer October 1999.

Bosak, Jon. "XML, Java and the Future of the Web" <http://metalab.unc.edu/pub/sun-info/standards/xml/why/xmlapps.htm>.

Bradley, Neil. The XML Companion. New York: Addison Wesley, 2000.

Burnard, Lou; "What is SGML and How Does it Help?" *XML Cover Pages,* December 1996 <http://http://www.oasis-open.org/cover/burnardw25-index.html> (May 3, 2000).

Busby, J. S., and Hutchinson, D. "The Practical Integration of Manufacturing Applications". Software-Practice and Experience. 22.2 (1992): 183-207.

Claben, Michel. "XML - the better HTML?" <www.webreference.com/xml/column1/index.html>.

Davis, Phil. "MES:Why, When How-Wow!" *Midrange ERP* <http://wwwibaset.com/Success/Client/Why_When_How/Why_When_how.htm>(Jan 24, 2000)

De Rose, Steven. The SGML FQA Book (Understanding the fOundation of HTML and XML). Kluwe Academic Publishers, 1997.

Duffner, Lynn. "Forecasting TO Meet the Demand for Cellular Service". Educational Society for Resource Management 8.12 (1998).

Fucher, Jim. "Alive and well". Manufacturing Systems September 1999:1a-14a.

Fulcher, Jim. "Global Solution" Manufacturing Systems September 1999: 23a-24a.

Gagnon, Gabrielle. "SQL: The Universal Database Language". PC Magazine November 1998: 285+.

Ganeshan, Ram. " An Introduction to Supply Chain Management". <http://silmaril.smeal.psu.edu/misc/supply_cahin_intro.html> (January 10, 2000).

Garikapati, Ramarao et al. " Induatry Specific Markup Languages for Intranet-based Decision Support Systems". IIE Solutions Conference. Cleveland, June 2000.

Gerson, Phil, et al. "Response to OMG Manufacturing Domain Task Force RFI-3: Manufacturing Execution Systems (MES)". *BOEING*, March 11, 1998 <www.boieng.com>.

Glushko, Robert, et al. "An XML Framework for Agent-based E-Commerce". Communications of the ACM 42.3 (1999): 106-114.

Goldfarb, Charles, and Prescod, Paul. The XML Handbook, 2nd Edition. New Jersey: Prentice Hall, 2000.

Goldfarb, Charles; The SGML Handbook; Oxford: Clarendon Press, 1990.

Goranson, H. T. The Agile Virutal Enterprise. Cases, Metrics, Tools. Connecticut: Quorum Books, 1999.
Goth, Greg. "XML Spces Duel Over E-Commerce". Computer April 1999: 17-19.

Greenbaum, Joshua. "The Origin and Future of ERP" *Origin ERP*. <www.erp-outsourcing.com>

Haggesty, Paul, and Seetharaman, Krishnan. "The benefits of CORBA-Based network management". Communications of the ACM 41.10 (1998).

Hamstra, Dirk. "XML and CORBA". Dr. Dobb's Journal November 1999: 98+.

Hardwick, Martin, et al. "Sharing Manufacturing Information in Virtual Enterprises". Communications of the ACM 39.2 (1996): 46-54.

Hennning, Michi "Binding, Migration, and Scalability in CORBA". Communications of the ACM 41.10 (1998).

Hewitt, David. "Distributed Computing in the Manufacturing Environment". Computer Integrated Manufacturing Systems: Selected Readings (1985): 173+.

Hicks, Donald a, and Stecke, Kathryn E. "The ERP maze: Enterprise Resource Planning and Other Produciton and Inventory Control Software". IIE Solutions 27.8 (1995): 12+.

Hill, Sidney. "A Net Gain". Manufacturing Systems November 1999: 34+.

Homer, Alex, et al. Professional Active Server Pages 3.0. Birminghan: WROX Press, 1999.

Jack, Hugh. "Corporate Communications" *Automated Manufacturing Engineer on a disk*. 1999. <http://claymore.engineer.gvsu.edu/eod/automate/integration/integration-2.html> (March 28, 2000).

Jones, Albert, and McLean, Charles. "A proposed Hierarchical Control Model for Automated Manufacturing Systems". Journal of Manufacturing Systems 5.1 (1986): 15-25.

Keller, Erik. "Lessons Learned". Manufacturing Systems November 1999: 44+.

Klapper, Norman. HI/MIS: High Impact Manifacturing Information Systems. Van Nostrand Reinhold, 1992.

Koch, Christopher, and Slater, Derek, and Baatz, E. "The ABCs of ERP". *Enterprise Systems Research Center*. <http://www.cio.com/forums/erp/edit/122299_erp.html> (Jan 11, 2000).

Labs, Wayne. "Standardized Software Drivers: A Plus to Some, Negative to Others". I&CS June 1991: 67+.

Lars , Marius, Garshol. "Introduction to XML". <http://www.stud.ifi.uio.no/~lmariusg/download/xml/xml_eng.html>.

Linthicum, David. "Multithreaded application Integration" Component Strategies 1.6 (1998).

Mahmoud, Qusay H. "Develop distributed Components with CORBA". Component Advisor January 2000: 32-34.

Manufacturing Executions Systems Association (MESA)< www.mesa.org>.

McClellan, Michael. Applying Manufacturing Execution Systems. Florida: St Lucie Press, 1997.

McClure, Steve. "Object Database vs. Object-relationa Databases" *International Data Corporation.* August 1997. <http://www.cai.com/products/jasmine/analyst/idc/14821E.htm> (March 13, 2000).

Metz, Peter. "Demystifying Supply Chain Management. Winter 1998. <http://www.econ.cbs.dk/organizations/logistik/images/myst.htm> (January 10, 2000).

MICO <www.icsi.berkeley.edu/~mico>.

Minnick, Chris. "An XML Toolkit". Software Development, October 1999: 46-51.

Mowbray, Malveau. CORBA Design Patterns. Wiley, 1997.

Murphy, Niall. "Introduction to CORBA for embedded systems". Embedded Systems Programming October 1998.

Oleson, John D. Pathways to Agility: Mass Cutomization in Action. New York: Wiley, 1998.

On track Manufacturing Execution System. http://www.ontrack-mes.com

Orfali, Robert, Harkey, Dan, Edwars. Instant CORBA. Wiley, 1997.

Orfali, Robert, Harkey, Dan, Maki, Krista. "Client-server systems in manufacturing". CS September, 1991

Otte, Randy, et al. Understanding CORBA. Prentice-Hall, 1996.

Parsaei, Hamid, and Jamshidi, Mohammad. Design and Implementation of Intelligent Manufacturing Systems. New Jersey: Prentice Hall, 1995.

Pompeii, John. "Programming with CORBA and DCOM". Byte April 1997.

Rathmill, K . Control and Programming on Advance Manufacturing. United Kingdom: IFS Publications, 1988.

Rehg, James. Computer-Integrated Manufacturing. New Jersey: Prentice Hall, 1994.

Rhoads, Michael. "Make Sure that the Enabler is Enabling" I&CS December 1991: 39-41.

Robinson, Phil. "Enterprise Resource Planning Survival Guide". <wysiwyg://257/htpp://www.bpic.com.uk/erp.htm> (Jan 18, 2000).

Roch, Arthur. "Flexible Machining in a Integrated System". Design and Analysis of Integrated Manufacturing Systems 1988:34-45.

San Jose State University. http://profdev.sjsu.edu

Scharpf, Sara. "ERP $ MES: Information Revolution Sparks an IS Evolution". PIMA's Papermaker. 81.3 (1999): 42-4.

Schmidt, Douglas. "Evaluating architectures for multithreaded object Request Brokers". Communications of the ACM 41.10 (1998).

Seetharaman, Krishnan. "The CORBA connection". Communications of the ACM 41.10 (1998).

Shankarnarayanan, S. "ERP Systems, Using IT to Gain a Competitive Advantage". *Query at Ask Jeeves* <www.askjeeves.com>.

Shiau, Jiun-Yan and Wolfe, Philip M. " An XML Solution to Supply Chain Information Integration". IIE Solutions Conference. Cleveland, June 2000.

Shrensker, Warren. "A Brief History of CIM". CIM Implementation Guide (1991): 7-9

Siegel, Jon. "OMG overview: CORBA and the OMA in enterprise computing". Communications of the ACM 41.10 (1998).

Siegel, Jon. CORBA Fundamentals and Programming Wiley, 1996.

Singh, Nanua. Systems Approach to Computer Integrated Design and Manufacturing New York: John Wiley, 1996.

Snyder, Merle. "Enterprise-wide Software can Add Meaning to CIM Data". Modern Plastics

Svenson, Margaret D., and Lozier, Barry. "MES Closes CIM Gap On The Plant Floor". INTEC September 1993: 37-41.

Taffe, Michael. "End of a Century-Start of New Era". The Enterprise 3.1 (1999): 80+

Van Herwijnen, Erci; Practical SGML; Kluwer Boston: Academic Publishers, 1994.

Vaughn, Richard C. Introduction to Industrial EngineeringIowa: The Iowa State University Press, 1967.

Vemadat, François. Enterprise Modeling and Integration: Principles and Applications. London: Chapman, 1996.

Vinoski, Steve. "New features for CORBA 3.0". Communications of the ACM 41.10 (1998).

Vogel, Andreas, et al. C++ Programming with CORBA. New York: Wiley, 1999.

Webopaedia; Computer Related Internet Dictionary. < www.webopedia.internet.com>

Word Wide Web Consortium. <www.w3.org/xml>.

9. APPENDIX

Next| Up| Previous|
Next: Separating client and server **Up:** Sample Program **Previous:** Standalone program

MICO application

Now we want to turn the standalone implementation from the previous section into a MICO application. Because CORBA objects can be implemented in different programming languages[3.2] the specification of an object's *interface* and *implementation* have to be separated. The implementation is done using the selected programming language, the interface is specified using the so called *Interface Definition Language (IDL)*. Basically the CORBA IDL looks like C++ reduced to class and type declarations (i.e., you *cannot* write down the implementation of a class method using IDL). Here is the interface declaration for our account object in CORBA IDL:

```
interface Account {
    void deposit (in unsigned long amount);
    void withdraw (in unsigned long amount);
    long balance ();
};
```

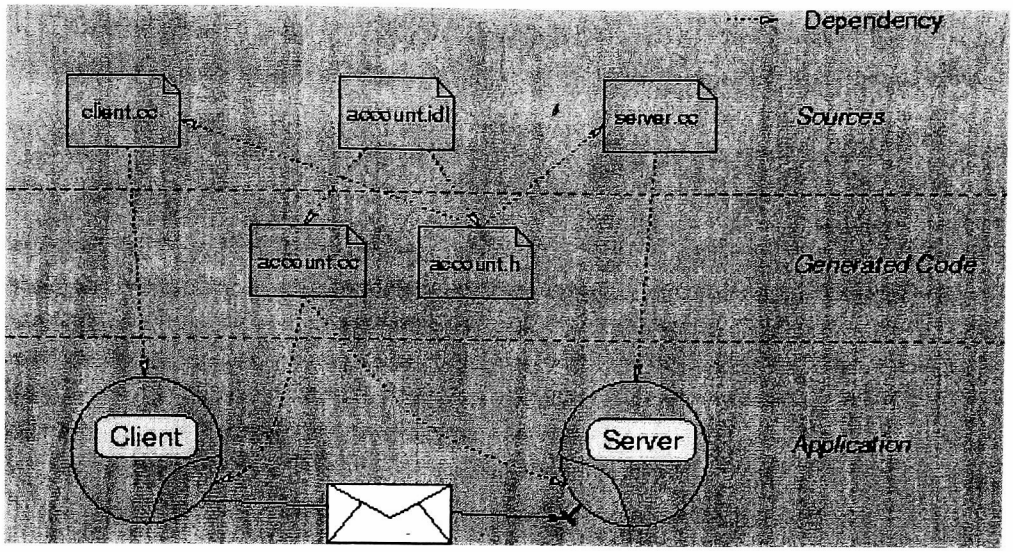
As you can see it looks quite similar to the class declaration in section 3.3.1. The `in` declarator declares `amount` as an input parameter to the `deposit()` and `withdraw()` methods. Usually one would save the above declaration to a file called `account.idl`.

The next step is to run this interface declaration through the *IDL compiler* that will generate code in the selected implementation programming language (C++ in our example). The MICO IDL compiler is called `idl` and is used like this:

```
idl account.idl
```

The IDL compiler will generate two files: `account.h` and `account.cc` (see figure 3.2). The former contains class declarations for the base class of the account object implementation and the stub class a client will use to invoke methods on remote account objects. The latter contains implementations of those classes and some supporting code. For each interface declared in an IDL-file, the MICO IDL compiler will produce three C++ classes[3.3].

Figure 3.2: Creation process of a MICO application.



The three classes are depicted in figure 3.3 between the two dashed lines. The class `Account` serves as a base class. It contains all definitions which belong to the interface `Account`, like local declarations of user defined data structures. This class also defines a pure virtual function for each operation contained in the interface. The following shows a bit of the code contained in class `Account`:

```
// Code excerpt from account.h
class Account : virtual public CORBA::Object {
    ...
public:
    ...
    virtual void deposit (CORBA::ULong amount) = 0;
    virtual void withdraw (CORBA::ULong amount) = 0;
    virtual CORBA::Long balance () = 0;
}
```

The class `Account_skel` is derived from `Account`. It adds a dispatcher for the operations defined in class `Account`. But it does not define the pure virtual functions of class `Account`. The classes `Account` and `Account_skel` are therefore abstract base classes in C++ terminology. To implement the account object you have to subclass `Account_skel` providing implementations for the pure virtual methods `deposit()`, `withdraw()` and `balance()`.

The class `Account_stub` is derived from class `Account` as well. In contrast to class `Account_skel` it defines the pure virtual functions. The implementation of these functions which is automatically generated by the IDL-compiler is responsible for the parameter marshalling. The code for `Account_stub` looks like this:

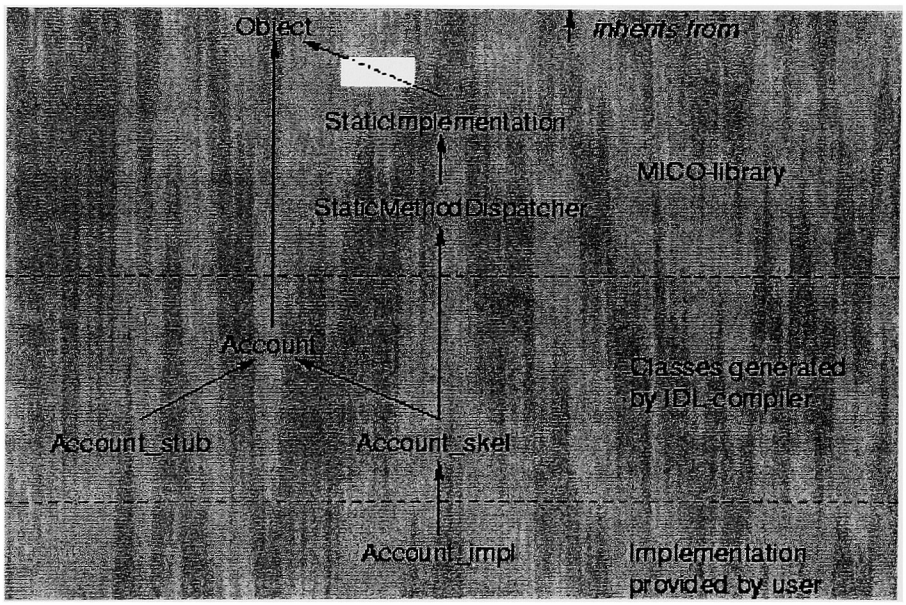
```
// Code excerpt from account.h and account.cc
class Account;
typedef Account *Account_ptr;

class Account_stub : virtual public Account {
    ...
public:
    ...
    void deposit (CORBA::ULong amount)
    {
        // Marshalling code for deposit
    }
```

```
    void withdraw (CORBA::ULong amount)
    {
       // Marshalling code for withdraw
    }
    CORBA::Long balance ()
    {
       // Marshalling code for balance
    }
}
```

This makes `Account_stub` a concrete C++ class which can be instantiated. The programmer never uses the class `Account_stub` directly. Access is only provided through class `Account` as will be explained later.

Figure 3.3: Inheritance relationship between stub- and skeleton classes.



It is worthwile to see where the classes `Account` and `Account_skel` are derived from. `Account` inherits from `Object`, the base class for all CORBA objects. This class is located in the MICO library. The more interesting inheritance path is for `Account_skel`. `Account_skel` inherits from `StaticMethodDispatcher`, a class located again in the MICO library. This class is responsible for dispatching a method invocation. It maintains a list of method dispatchers[3.4]. The class `StaticMethodDispatcher` inherits from `StaticImplementation`. This class mirrors the behaviour of the *dynamic skeleton interface* (DSI), but is more efficiently designed.

Up until now we have written the interface of an account object using CORBA IDL, saved it as `account.idl`, ran it through the IDL compiler which left us with two files called `account.cc` and `account.h` that contain the class declarations for the account implementation base class (`Account_skel`) and the client stub (`Account_stub`). Figure 3.2 illustrates this. What is left to do is to subclass `Account_skel` (implementing the pure virtual methods) and write a program that uses the bank account. Here we go:

```
1: #include "account.h"
2:
3: class Account_impl : virtual public Account_skel
```

```
 4: {
 5: private:
 6:   CORBA::Long _current_balance;
 7:
 8: public:
 9:   Account_impl()
10:   {
11:     _current_balance = 0;
12:   };
13:   void deposit( CORBA::ULong amount )
14:   {
15:     _current_balance += amount;
16:   };
17:   void withdraw( CORBA::ULong amount )
18:   {
19:     _current_balance -= amount;
20:   };
21:   CORBA::Long balance()
22:   {
23:     return _current_balance;
24:   };
25: };
26:
27:
28: int main( int argc, char *argv[] )
29: {
30:   // ORB initialization
31:   CORBA::ORB_var orb = CORBA::ORB_init( argc, argv, "mico-local-orb" );
32:   CORBA::BOA_var boa = orb->BOA_init( argc, argv, "mico-local-boa" );
33:
34:   // server side
35:   Account_impl* server = new Account_impl;
36:   CORBA::String_var ref = orb->object_to_string( server );
37:   cout << "Server reference: " << ref << endl;
38:
39:   //--------------------------------------------------------------------
40:
41:   // client side
42:   CORBA::Object_var obj = orb->string_to_object( ref );
43:   Account_var client = Account::_narrow( obj );
44:
45:   client->deposit( 700 );
46:   client->withdraw( 250 );
47:   cout << "Balance is " << client->balance() << endl;
48:
49:   // We don't need the server object any more. This code belongs
50:   // to the server implementation
51:   CORBA::release( server );
52:   return 0;
53: }
```

Lines 3-25 contain the implementation of the account object, which is quite similar to the implementation in section 3.3.1. Note that the class `Account_impl` inherits the from class `Account_skel`, which contains the dispatcher for this interface, via a virtual public derivation. Although the keyword `virtual` is not required in this case, it is a good practise to write it anyway. This will become important when interface inheritance is discussed in section 5.5.

The `main()` function falls into two parts which are seperated by the horizontal line (line 39): Above the separator is the server part that provides an account object, below the line is the client code which invokes methods on the account object provided by the server part. Theoretically the two parts could be moved to two seperate programs and run on two distinct machines and almost nothing had to be changed in the code. This will be shown in the next section.

In line 32 the MICO initialization function is used to obtain a pointer to the *Object Request Broker (ORB)* object--a central part of each CORBA implementation. Among others the ORB provides methods to convert object references into a string representation and vice versa. In line 35 an account object called `server` is instantiated. Note that it is not permitted to allocate CORBA objects on the run-time stack. This is because the CORBA standard prescribes that every object has to be deleted with a special function called `CORBA::release()`. Automatic allocation of an object would invoke its destructor when the program moves out of scope which is not permissible. In our little sample program the server object is deleted explicitly in line 51.

In line 36 the ORB is used to convert the object reference into a string that somehow has to be transmitted to the client (e.g., using Email, a name service or a trader). In our example client and server run in the same address space (i.e. the same process) so we can turn the string into an object reference back again in line 42. Line 43 uses the `Account::_narrow()` method to downcast the object reference to an `Account_var`. The rest of `main()` just uses the account object which was instantiated in line 35.

`Account_var` is a smart pointer to `Account` instances. That is an `Account_var` behaves like an `Account_ptr` except that the storage of the referenced object is automatically freed via the aforementioned `release()` function when the `Account_var` is destroyed. If you use `Account_ptr` instead you would have to use `CORBA::release()` explicitly to free the object when you are done with it (*never* use `delete` instead of `CORBA::release()`).

Assuming the above code is saved to a file called `account_impl.cc` you can compile the code like this[3.5]:

```
mico-c++ -I. -c account_impl.cc -o account_impl.o
mico-c++ -I. -c account.cc -o account.o
mico-ld -I. -o account account_impl.o account.o -lmico2.3.3
```

This will generate an executable called `account`. Running it produces the following output:

```
Server reference: IOR:010000001000000049444c3a4163636f756e743a312e3\
000020000000000000030000000010100001300000752d6d61792e7468696e6b6f\
6e652e636f6d00007b0900000c000000424f410a20b0530000055f0301000000240\
000000100000001000000010000001400000001000000010001000000000090101\
0000000000
Balance is 450
```

You can find the source code for this example in the `demo/boa/account` directory within the MICO source tree. Note that the IOR may look different on different systems. This is because it contains information which depend on the hostname, port number and object ID for the server object among other things. There is a tool called *iordump* (see directory `mico/tools/iordump`) which shows the content of the IOR. Feeding the IOR above into *iordump* yields the following output:

```
    Repo Id:  IDL:Account:1.0

IIOP Profile
    Version:  1.0
    Address:  inet:u-may.thinkone.com:2427
   Location:  iioploc://u-may.thinkone.com:2427/BOA%0a%20%b0S%00%00%05%5f%03
        Key:  42 4f 41 0a 20 b0 53 00 00 05 5f 03              BOA. .S..._.

Multiple Components Profile
 Components:  Native Codesets:
              normal. ISO 8859-1:1987; Latin Alphabet No. 1
                wide: ISO/IEC 10646-1:1993; UTF-16, UCS Transformation Format\
                                                16-bit form
```

```
Key:  00
```

Next Up Previous

Next: Separating client and server **Up:** Sample Program **Previous:** Standalone program
MICO documentation

Next| Up| Previous|
Next: MICO application **Up:** Sample Program **Previous:** Sample Program

Standalone program

Imagine a bank which maintains accounts of its customers. An object which implements such a bank account offers three operations[3.1]: *deposit* a certain amount of money, *withdraw* a certain amount of money, and an operation called *balance* that returns the current account balance. The state of an account object consists of the current balance. The following C++ code fragment shows the class declaration for such an account object:

```
class Account {
    long _current_balance;
public:
    Account ();
    void deposit (unsigned long amount);
    void withdraw (unsigned long amount);
    long balance ();
};
```

The above class declaration describes the *interface* and the *state* of an account object, the actual *implementation* which reflects the behavior of an account, is shown below:

```
Account::Account ()
{
    _current_balance = 0;
}
void Account::deposit (unsigned long amount)
{
    _current_balance += amount;
}
void Account::withdraw (unsigned long amount)
{
    _current_balance -= amount;
}
long Account::balance ()
{
    return _current_balance;
}
```

Here is a piece of code that makes use of a bank account:

```
#include <iostream.h>

int main (int argc, char *argv[])
{
    Account acc;

    acc.deposit (700);
    acc.withdraw (250);
    cout << "balance is " << acc.balance() << endl;

    return 0;
}
```

Since a new account has the initial balance of 0, the above code will print out ``*balance is 450*''.

Transcription

Manassas junction
Oct. 8th 1861

Dear Cousin

I write afew lines this
morning to inform you that I am well
at this time and hopeing that it
may find you all injoying the same
blesing, the health of our company
is better at this time than it has
bin for some time,

I have no news of intrust to write
to you, it is thought that we
will have a battle in a few days, its
reported that thay was fighting
yesterday at fawls Church I dont [know] weth
er it was so or not, one of the Dan
ville Grays was upto see us last night
he said the yankees was in four
miles of them thay are stationed at
Farfax Court House six miles a head of
us, it is thought that we will
have a verry hard battle when it
does come off, I received a letter from
Addie [add note 1] last eavning it [[unclear:]] afforded me
great pleasure to hear that he was
improveing so fast,

I will ad no more at [unclear: present] so good bye

[Page 2]

write soon to your affectionate Cousin

James Booker

To Miss C. U. Blair

Notes

[1] "Addie" probably refers to Drury Addison Blair (1839-1864), the Bookers' cousin. Blair joined Company D when it was formed in May of 1861, but was discharged due to chronic bronchitis in August of 1861 (Gregory 81). See James Booker's letter of July 14, 1861, in which "A. Blair" includes a postscript to Chloe Unity Blair.

Tagged Version of the Letter

```
<TEI.2>

 <teiHeader> [TEI Header goes here] </teiHeader>

 <text id="Boo1j08">
  <body>
   <div1 type="letter" n="1861-10-08">
    <pb n="1">
     <opener>

        <dateline>

           <name type="place">Manassas
               <orig reg="Junction">junction</orig>
           </name>

           <date n="1861-10-08">
               <abbr expan="October">Oct.</abbr>
               8
               <hi rend="superscript"> th </hi>
               1861
           </date>

        </dateline>

        <salute>Dear Cousin</salute>

     </opener>

     <p>
       I write <orig reg="a few">afew</orig> lines this <lb>
       morning to inform you that I am well <lb>
       at this time and <orig reg="hoping">hopeing</orig> that it <lb>
       may find you all <orig reg="enjoying">injoying</orig> the same <lb>
       <orig reg="blessing. The">blesing, the</orig> health of our company<lb>
       is better at this time than it has <lb>
       <orig reg="been">bin</orig> for some <orig reg="time.">time,</orig>
     </p>

     <p>
        I have no news of <orig reg="interest">intrust</orig> to write <lb>
        to <orig reg ="you. It">you, it </orig> is thought that we <lb>
        will have a battle in a few <orig reg="days. It's">days, its</orig><lb>
        reported that <orig reg="there">thay</orig> was fighting <lb>
        yesterday at
            <name type="place">
                  <orig reg="Falls Church.">fawls Church </orig>
            </name>
        <orig reg="don't">dont</orig> <add n="editor">know</add>
        <orig reg="whether">weth <lb>
            er </orig> it was so or
        <orig reg="not. One">not, one</orig> of the <orig reg="Danville">Dan<lb>
        ville</orig> Grays was <orig reg="up to">upto</orig> see us last
        <orig reg="night.">night</orig> <lb>
        <orig reg="He">he</orig> said the yankees was in four <lb>
        miles of <orig reg="them.">them</orig>
        <orig reg="They">thay</orig> are stationed at <lb>
```

```
        <name type="place">
            <orig reg="Fairfax">Farfax</orig> Court House
        </name>
        six miles <orig reg="ahead">a head</orig> of <lb>
        <orig reg="us. It">us, it</orig> is thought that we will <lb>
        have a <orig reg="very">verry</orig> hard battle when it <lb>
        does come <orig reg="off.">off,</orig> I received a letter from <lb>

        <name type="person">Addie
        </name>

        <note target="n1">[1]
        </note>

        last <orig reg="evening. It">eavning it</orig>

        <del><unclear></unclear></del>
        afforded me <lb>
        great pleasure to hear that he was <lb>
        <orig reg="improving">improveing</orig> so
        <orig reg="fast.">fast,</orig>
      </p>

      <closer>
       <salute>
          I will <orig reg="add">ad</orig> no more at
          <unclear reason="under folded page edge">present</unclear>
          so <orig reg="goodbye.">good bye</orig>

        <pb n="2">
        <orig reg="Write">write</orig> soon to your affectionate
        Cousin
       </salute>

       <signed>
         <name type="person">
            James Booker
         </name>
       </signed>
       <seg type="recipient">To Miss C. U. Blair</seg>
      </closer>
    </div1>
  </body>

  <back>
    <div1 type="notes">
      <head>Notes</head>
            <note id="n1">
              [1] "Addie" probably refers to Drury Addison Blair   (1839-
              1864),the Bookers' cousin. Blair joined Company D when it was
              formed in May of 1861, but was discharged due to chronic
              bronchitis in August of 1861 (Gregory 81). See James Booker's
              letter of July 14, 1861, in which "A. Blair" includes a postscript
              to Chloe Unity Blair.
            </note>
    </div1>
   </back>
  </text>
 </TEI.2>
```

List Of Courses Offered

EIEI 787 Systems Optimization(for PD21)
EIEI 630/729 Computer Integrated Manufacturing
EIEI 775: Data Structures Using C
Last modified: April 5, 2000

```
<html>
 <head>
  <title>courses Listing</title>
  <link rel="stylesheet" href="styles.css">
  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
  <meta name="Microsoft Theme" content="copy-of-sumi-painting 011, default">
 </head>
 <body background="_themes/copy-of-sumi-painting/sumtextb.jpg" bgcolor="#ffffff" text="#000000" link="#800080" vlink="#8000
80" alink="#800080">
   <table border="0" align="center" cellpadding="1" width="400">
    <tr bgcolor="#660066">
      <td height="25"><!--mstheme-->
        <font face="Verdana, Arial, Helvetica">
         <div align="center">
          <b>
           <font color="#ffffff" face="Arial, Helvetica" size="4">
            List Of Courses Offered
           </font>
          </b>
         </div>
        </font>
      </td>
    </tr>

  <tr bgcolor="#cccccc">
   <td>
    <DIV align="center">
     <DIV align=left>
       <font color="#ffffff">
            <font face="" size    ="4" color="#660066">
                    
                                          <A href="sysopt/index.html">
                                           EIEl 787 Systems Optimization(for P21)
                                          </A>
                                          <BR>
            </font>
       </font>

         <FONT size=3>
          <FONT face="Times New Roman">
           <font color="#660066">
            <A href="cim99/index.html">
             EIEl 630/729 Computer Integrated Manufacturing
            </A>
           </font>
          </FONT>
         </FONT>

         <FONT size=3>
          <FONT face="Times New Roman">
           <BR>
           <font color="#660066">
                      
            <A href="datastr/index.html">
             EIEl 775: Data Structures Using C
            </A>
            <BR>
                
            Last modified: April 5, 2000
           </font>
          </FONT>
         </FONT>
     </DIV>
    </DIV>
   </td>
  </tr>

  <tr bgcolor="#660066">
    <td height="25">
    </td>
  </tr>

    </table>
 </body>
</HTML>
```

So away with HTML?

Certainly not. Its ubiquitous use on the millions of documents available around the planet will have it live and prosper for years to come. The W3C is currently working on XHTML, a definition of HTML in strict XML, so that browser parsers can be simplified and the vast array of XML tools that become available will also be useful to HTML developers. The proper use of style sheets with its separation of content and style will open up a whole new world of document and data exchange, especially in a fully automated fashion.

Conclusion

XML fixes some of the problems that were created with HTML, but has a much broader scope to allow for a real comparison. XML and the W3C-created standards on top of it for linking, style sheets and document access will in the near future provide an equivalent platform for displaying documents, while opening up opportunities for much simpler browsers, machine-readable documents, and custom tag vocabularies, both standardized and user-defined. Universal data exchange between humans and applications, as well as between heterogeneous applications, will become a reality soon. Good times to be an HTML/XML developer!

Practice what you preach, or: Eat your own dog food

...was my goal, so I wrote this article in XML as its source form , and used an XSL stylesheet to turn it into the HTML you have been reading just now. Good to know it works not only on paper...and thanks for reading! I appreciate your feedback via email to me .

URL: http://www.WebReference.com/xml/column1/page4.html

Created: Nov. 20, 1999
Revised: Dec. 31, 1999

XML - the better HTML?

Let's face it: HTML has already passed the logical point for its further development. The W3C will finally declare version 4.0 the last in history, after a serious of questionable innovations, all the way from the tag in version 2, over frames in 3.2, to more nonsense in 4.0. While HTML has brought a tidal wave of information sharing to the world via the Web, it brought Web developers nothing but grief: Bloated browsers with incompatible implementations of slow and sometimes silly standards. But fear not, XML comes to the rescue - or, then again, not?

So is XML better?

To be honest this question has no real answer: XML is a meta language, meaning a language for defining other languages, while HTML by itself is a more or less well-defined language. XML stands for eXtensible Markup Language, which is actually a bit of a misnomer as it should actually mean extensible *Meta* Language. The easiest way to understand the difference is to note that XML by itself does not define any tags, it only describes a way of defining your own set of tags and attributes, hence the name extensible. HTML in contrast has a fixed set of tags, and their meaning is defined in the W3C standards specifications or the implementation of a particular browser, whichever came first. So in directly comparing XML and HTML one would compare apples and oranges.

So what is wrong with HTML?

Let's look at a specific example. The following fragment of HTML shows a listing of a shopping cart containing two products, as it might appear on any of your favorite shopping Web sites:

Your shopping cart contains:

Pen	3	3.99
Pencil	2	2.95

See it in the browser .

There are many good things to be said about HTML:

- It is relatively simple, therefore quick and easy to learn.
- It is now pervasive in the Internet space, not least because of its simplicity.
- It can be viewed with minimal client requirements, namely a browser.
- It is well suited for describing the visual appearance of a human-readable document, including text and images.

But there are also some shortcomings:

- It mixes data structure, e.g. the articles in the shopping cart in our example above, with presentation instructions such as a table border of one pixel width, and a row background color of red.
- It does not identify the data elements, so the information that pens and pencils are articles, and that 3.99 and 2.95 are their respective prices, is lost.
- It uses a fixed set of well-defined tags, it is not extensible to allow for user-defined tags. There are no tags for describing vector graphics in HTML, for example.
- It is good enough for humans, i.e. to be displayed in a browser, but not good enough for use by machines: Could you easily write a program to calculate the total order price for our shopping cart above?

URL: http://www.WebReference.com/xml/column1/index.html
Created: Nov. 20, 1999

Revised: Dec. 09, 1999

```
<?xml version="1.0"?>
<?xml-stylesheet href="html.doc.xsl" type="text/xsl"?>
<document>
<title>XML   the better HTML?</title>
<author>Michael Cla&#223;en</author>
<section>
<p>Let's face it: HTML has already passed the logical point for its further developmen
The W3C will finally declare version 4.0 the last in history, after a serious of
questionable innovations, all the way from the &lt;blink&gt; tag in version 2, over fr
to more nonsense in 4.0. While HTML has brought a tidal wave of information sharing to
world via the web, it brought Web developers nothing but grief: Bloated browsers with
incompatible implementations of slow and sometimes silly standards.
But fear not, XML comes to the rescue   or, then again, not?</p>
</section>
<section>
<title>So is XML better?</title>
<p>To be honest this question has no real answer: XML is a meta language, meaning a la
for defining other languages, while HTML by itself is a more or less well-defined lang
XML stands for eXtensible Markup Language, which is actually a bit of a misnomer as it
should actually mean extensible <emph>Meta</emph> Language. The easiest way to underst
is to note that XML by itself does not define any tags, it only describes a way of def
your own set of tags and attributes, hence the name extensible. HTML in contrast has a
set of tags, and their meaning is defined in the W3C standards specifications and/or t
implementation of a particular browser, whichever came first. So in directly comparing
and HTML one would compare apples and oranges.</p>
</section>
<section>
<title>So what is wrong with HTML?</title>
<p>Let's look at a specific example. The following fragment of HTML shows a listing of
shopping cart containing two products, as it might appear on any of your favorite shop
Web sites:</p>
<code source="html" target="html" link="inline" href="xml1a.htm"><![CDATA[
<HTML>
  <BODY>
    <H3>Your shopping cart contains:</H3>
    <TABLE BORDER="1">
      <TR BGCOLOR="#FF0000"><TH>Article</TH><TH>Qty</TH><TH>Price</TH></TR>
      <TR><TD>Pen</TD><TD>3</TD><TD>3.99</TD></TR>
      <TR><TD>Pencil</TD><TD>2</TD><TD>2.95</TD></TR>
    </TABLE>
  </BODY>
</HTML>
]]>
</code>
<p><doclink href="xml1a.htm">See it in the browser</doclink>.</p>
<p>There are many good things to be said about HTML:</p>
<list>
<item>It is relatively simple, therefore quick and easy to learn.</item>
<item>It is now pervasive in the Internet space, not least because of its simplicity.<
<item>It can be viewed with minimal client requirements, namely a browser.</item>
<item>It is well suited for describing the visual appearance of a human-readable docum
including text and images.</item>
</list>
<p>But there are also some shortcomings:</p>
<list>
<item>It mixes data structure, e.g. the articles in the shopping cart in our example a
with presentation instructions such as a table border of one pixel width, and a row
background color of red.</item>
<item>It does not identify the data elements, so the information that pens and pencils
articles, and that 3.99 and 2.95 are their respective prices, is lost.</item>
<item>It uses a fixed set of well-defined tags, it is not extensible to allow for user
tags. There are no tags for describing vector graphics in HTML, for example.</item>
<item>It is good enough for humans, i.e. to be displayed in a browser, but not good en
for use by machines: Could you easily write a program to calculate the total order pri
for our shopping cart above?</item>
```

```
</list>
</section>
<section>
<title>So what exactly is XML?</title>
<p>XML is a markup language derived from SGML, the Standard Generalized Markup Languag
whose predecessor GML was invented by IBM in the 60s for describing documents in a
device-independent fashion. XML documents are text-based, as opposed to binary like e.
COM or CORBA.  XML documents contain text, and tags identifying structures within the
just like HTML. Unlike HTML, XML allows custom tags and attributes. This is like SGML,
but much simpler and without the technical complexities of its ancestor.</p>
<p>The W3C recommends a set of standards based on XML for common topics like document
linking and style sheets, in order to provide the basis for applications to effectivel
share data.</p>
<p>Due to its flexible nature, XML is useful not only in a browser for human consumpti
but also for application-to-application data exchange, such as business-to-business tr
, for example to facilitate supply chains and other business communication. Unlike ear
business-to-business initiatives like EDI (Electronic Data Interchange) XML does not
require a costly value-added network (VAN) or a company Wide Area Network (WAN), becau
just like HTML XML can be transported over the Internet via HTTP. It can thereby take
advantage of the whole Internet infrastructure, such as Web servers and SSL security.<
<p>XML documents may contain meta data, i.e. data describing itself, and are hierarchi
structured. XML exceeds HTML limitations in that it is extensible, but its structure i
fact much more rigid than HTML. All documents are supposed to be well-formed, and some
be validated against a document type definition. More on this in the next installment
this column, let us get back to our shopping cart example for now. What does it look l
in XML?</p>
<code source="xml" target="xml" link="window" href="xml1a.xml"> <![CDATA[
<?xml version="1.0"?>
<shoppingcart>
  <item>
    <article>Pen</article><quantity>3</quantity><price unit="USD">3.99</price>
  </item>
  <item>
    <article>Pencil</article><quantity>2</quantity><price unit="USD">2.95</price>
  </item>
</shoppingcart>
]]>
</code>
<p>The first line declares the document to be an XML document, following the rules of
1.0 version of the XML standards definition. Next comes the root element &lt;shoppingc
containing two &lt;item&gt;s, one for our pens, and another one for our pencils. Each
wraps the article name, its ordered quantity and the item price. Simple to read and
understand for humans, and also easy to process for machines.</p>
</section>
<section>
<title>XML rules</title>
<p>XML consists of only a few constructs, elements and attributes. Elements are pairs
start and end tags, optionally with some text between them, like in &lt;article&gt;Pen
Attributes are name/value pairs that can be added to a start tag as in &lt;price unit=
<p>Some simple rules must be followed for constructing XML documents:</p>
<list>
<item>Match every start tag with an end tag, on the same hierarchy level. The XML pars
complain about missing or wrongly nested end tags, unlike an HTML parser which will tr
correct the error by himself.</item>
<item>Have exactly one root element, like the &lt;shoppingcart&gt;s element in our exa
<item>Enclose attribute values in quotes.</item>
<item>Quote the few special characters and keywords that exist when they appear in the
</list>
<p>This is even simpler than HTML for constructing documents, don't you think?</p>
</section>
<section>
<title>But HTML looks much better...</title>
<p>... at least in my browser, you could say. This is because the XML in our example d
not contain any style information, so even an XML-enabled browser like Internet Explor
can only render a rather boring-looking tree-style representation of the XML document.
```

```
style information needs to be supplied by a second XML document, expressed in the eXte
Stylesheet Language XSL, so that it looks like this:</p>
<p>Looks quite similar to the HTML at the beginning, right? And it was simple, too. Ju
added the second line to the XML file, et voila:</p>
<code source="xml" target="xml" link="window" href="xml1b.xml"> <![CDATA[
<?xml version="1.0"?>
<?xml-stylesheet href="xml1a.xsl" type="text/xsl"?>
<shoppingcart>
  <item>
    <article>Pen</article><quantity>3</quantity><price unit="USD">3.99</price>
  </item>
  <item>
    <article>Pencil</article><quantity>2</quantity><price unit="USD">2.95</price>
  </item>
</shoppingcart>
]]>
</code>
<p>Here is the accompanying XSL stylesheet:</p>
<code href="xml1a.xsl"> <![CDATA[
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<xsl:template match="/">
<HTML>
  <HEAD>
    <TITLE>Your shopping cart</TITLE>
  </HEAD>
  <BODY>
    <H3>Your shopping cart contains:</H3>
    <TABLE BORDER="1">
      <TR BGCOLOR="yellow"><TH>Article</TH><TH>Qty</TH><TH>Price</TH></TR>
      <xsl:for-each select="shoppingcart/item">
        <TR><TD>
          <xsl:value-of select="article" />
        </TD><TD>
          <xsl:value-of select="quantity" />
        </TD><TD>
          <xsl:value-of select="price" />
        </TD></TR>
      </xsl:for-each>
    </TABLE>
  </BODY>
</HTML>
</xsl:template>
</xsl:stylesheet>
]]>
</code>
<p>We will go into much more detail of XSL in a later installment of this column, but
those who cannot wait here comes an explanation of this particular style sheet:</p>
<p>XSL is a mix of a rules-based pattern matcher, a template apply mechanism and an
imperative programming language. The match instruction in line 3 matches the document
the top-most element of our document. The for-each loops over all item elements inside
shoppingcart element, and the value-of lines extract the values out of the document. A
other data, HTML text and tags in our case, are copied verbatim to the output document
The result matches exactly that of the HTML version of our shopping cart, shown above.
Since the part of XSL that defines document layout, the XSL flow and formatting object
(FO), is still missing, our only chance to get a visual representation of XML currentl
to create an HTML document using the already defined part of XSL for transformations,
XML browsers based on FO will surely appear after the standard is appropriately drafte
</section>
<section>
<title>But HTML has scripting for dynamic behavior...</title>
<p>The XML parser builds an in-memory tree structure of the document, the same way an
parser does. This tree can be accessed and manipulated using your favorite scripting l
like you are used to with the HTML document object model. In fact the W3C defines the
on several levels, for both HTML and XML. The style sheet language XSL provides its ow
```

```
of scripting capabilities as well as escapes to existing scripting languages such as
Javascript and VBScript.</p>
</section>
<section>
<title>So away with HTML?</title>
<p>Certainly not. Its ubiquitous use on the millions of documents available around the
planet will have it live an prosper for years to come. The W3C currently works on XHTM
definition of HTML in strict XML, so that browser parsers can be simplified and the va
array of XML tools that become available are also useful to HTML developers. The prope
of style sheets with its separation of content and style will open up a whole new worl
document and data exchange, especially in a fully automated fashion.</p>
</section>
<section>
<title>Conclusion</title>
<p>XML fixes some of the problems that were created with HTML, but has a much broader
to allow for a real comparison. XML and the W3C-created standards on top of it for lin
style sheets and document access will in the near future provide an equivalent platfor
displaying documents, while openíng up opportunities for much simpler browsers,
machine-readable documents, and custom tag vocabularies, both standardized and user-de
Universal data exchange between humans and applications, as well as between heterogene
applications, will become a reality soon. Good times to be an HTML/XML developer!</p>
</section>
<section>
<title>Eat your own dogfood...</title>
<p>...was my goal, so I wrote this article in XML as its <doclink href="xml1html.doc.x
source form</doclink>, and used an <doclink href="html.doc.xsl">XSL stylesheet</doclin
to turn it into the HTML you have been reading just now.
Good to know it not only works on paper...and thanks for reading! I would appreciate y
feedback via email to <email href="mailto:mclassen@internet.com">me</email>.</p>
</section>
</document>
```

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <HTML>
      <HEAD>
        <TITLE><xsl:value-of select="document/title"/></TITLE>
      </HEAD>
      <BODY>
        <H1><xsl:value-of select="document/title"/></H1>
                <EM>by <xsl:value-of select="document/author"/></EM>
        <xsl:apply-templates select="document/section"/>
      </BODY>
    </HTML>
  </xsl:template>

  <xsl:template match="section">
    <DIV>
      <H2><xsl:value-of select="title"/></H2>
      <xsl:apply-templates />
    </DIV>
  </xsl:template>

  <xsl:template match="p">
    <P><xsl:apply-templates /></P>
  </xsl:template>

  <xsl:template match="code">
    <PRE><xsl:apply-templates /></PRE>
  </xsl:template>

  <xsl:template match="list">
    <UL>
      <xsl:for-each select="item">
        <LI><xsl:apply-templates /></LI>
      </xsl:for-each>
    </UL>
  </xsl:template>

  <xsl:template match="emph">
    <EM><xsl:apply-templates /></EM>
  </xsl:template>

  <xsl:template match="strong">
    <STRONG><xsl:apply-templates /></STRONG>
  </xsl:template>

  <xsl:template match="email">
    <A><xsl:attribute name="HREF"><xsl:value-of select="@href"/></xsl:attribute><xsl:a
  </xsl:template>

  <xsl:template match="doclink">
    <A><xsl:attribute name="HREF"><xsl:value-of select="@href"/></xsl:attribute><xsl:a
  </xsl:template>

  <xsl:template match="text()"><xsl:value-of /></xsl:template>

</xsl:stylesheet>
```