Fall 1-31-2000

Integration of multi lifecycle assessment and design for environment database using relational moddel concepts

Bhagyashree Suratran
New Jersey Institute of Technology

ABSTRACT

INTEGRATION OF MULTI LIFECYCLE ASSESSMENT AND DESIGN FOR ENVIRONMENT DATABASE USING RELATIONAL MODEL CONCEPTS

by

Bhagyashree Suratran

Multi-lifecycle Assessment (MLCA) systematically considers and quantifies the consumption of resources and the environmental impact associated with a product or process. Design challenges posed by a multi-lifecycle strategy are significantly more complex than traditional product design. The designer must look forward in time to maximize the product's end-of-life yield of assemblies, parts and materials while looking backward to the world of existing products for feedstock sources for the current design. As MLCA and DFE share some common data items, such as, part geometry, material and manufacturing process, it is advantageous to integrate the database for MLCA and DFE. The integration of CAD/DFE and MLCA database will provide not only to designers but also for demanufacturer and MLCA analyst to play an active role in achieving the vision of sustainability.

The user of MLCA software has to provide a significant amount of information manually about a product for which the environmental burdens are being analyzed, which is an error prone activity. To avoid the manual work and associative problems, a MLCA-CAD interface has been developed to programmatically populate the MLCA database by using the Bill of Material (BOM) information in the CAD software. This MLCA-CAD interface provides a flow of information from design software (DFE/CAD) to MLCA software.

INTEGRATION OF MULTI LIFECYCLE ASSESSMENT AND DESIGN FOR ENVIRONMENT DATABASE USING RELATIONAL MODEL CONCEPTS

by
Bhagyashree Suratran

A Master's Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Masters of Science in Computer Science

Department of Computer and Information Science

January 2000

APPROVAL PAGE

INTEGRATION OF MULTI LIFECYCLE ASSESSMENT AND DESIGN FOR ENVIRONMENT DATABASE USING RELATIONAL MODEL CONCEPTS

Bhagyashree Suratran

Dr. Franz Kurfess, Thesis Advisor — Date
Director, Software Engineering Laboratory, NJIT
Associate Director, Electronic Enterprise Engineering, NJIT
Associate Professor of Computer Science, NJIT

Dr. Reggie Caudill, Thesis Co-Advisor — Date
Executive Director of Multi-lifecycle Engineering Research Center, NJIT
Professor of Industrial and Manufacturing Engineering, NJIT

Dr. Qian Hong Lue, Committee Member — Date
Assistant Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author: Bhagyashree Suratran

Degree: Masters of Science in Computer Science

Date: January 2000

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2000

- Bachelor of Science in Mechanical Engineering,
  Government College of Engineering, Pune, India, 1996

Major: Computer Science

To my beloved husband and my parents

ACKNOWLEDGMENT

I would like to express my deepest appreciation to Dr. Franz Kurfess, who not only served as my thesis advisor, but also constantly gave me support, valuable resources and insights.

I would also like to express my sincere gratitude to Dr. Regiee Caudill, who not only helped as my thesis co-advisor, but also gave me encouragement and reassurance.

Special thanks to Dr. Liu for her constructive comments and recommendations and actively participating in my committee.

I would like to take this opportunity to thank Ms. Elizabeth McDonnell, Ms Lori, Ms. Toia Moore and the staff at the Multi-Lifecycle Engineering Research Center (MERC) for their continuous motivation and suggestions.

Finally, I thank my friends and individuals not specifically delineated here who assisted me in this research.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 Background Information

Industrial growth and development seeks to meet present needs of society without compromising the ability of future generations to satisfy their own needs. In order to sustain further development, it is very essential to become more environmentally conscious in the development, manufacturing, use, and post-life treatment of products [10].

Over the last decade, America's manufacturing industry has struggled to achieve a balance between economic security and environmental responsibility. Government regulations are moving in the direction of "life-cycle accountability", the notion that a manufacturer is responsible not only for direct production impacts, but also for impacts associated with product inputs, use, transport, and disposal. (For example, Product 'Take-back' laws in European communities, requite manufactures to recover and recycle their discarded products.) In the USA, businesses are participating in voluntary initiatives, which contain Lifecycle Assessment (LCA) and product stewardship components. These include, for example, ISO 14000 and the Chemical Manufacturers Association's *Responsible Care* Program, both of which seek to foster continuous improvement through better environmental management systems.

Consumer electronics, computers, and household appliances contribute significantly to the environmental burden placed on the municipalities across the nation. The National Academy of Science reported that 94 percent of all natural resources from the earth enter the waste stream within months [3]. If discarded products and waste

streams can be recovered and reengineered into valuable feed streams, then sustainability can be achieved. For achieving this, a new approach, Multi-Lifecycle Engineering (MLCE) is necessary, which takes a system perspective and considers fully the potential of recovering and reengineering of materials and components from one product to create another, not just once, but many times [10].

1.2Aims and Objectives

Multi-lifecycle Assessment (MLCA) systematically considers and quantifies the consumption of resources and the environmental impact associated with a product or process. By considering the entire lifecycle and the associated environmental burdens, MLCA identifies opportunities to improve environmental performance, and reduce resource depletion.

A software tool, which the Multi-Lifecycle Engineering Research Center, at NJIT, is developing, will support all MLCA activities [17]. It is necessary to develop a flexible, extensible database schema which provides robust infrastructure for the entire MLCA and Design For Environment (DFE) project and provide a way to write extensible applications which have their own development cycle. Hence the design of database is extremely important because it supports the integrity, availability, standards based interface and the repository to various components in the MLCA Tool.

Thus the main objective for this thesis can be listed as follows:

1. To provide a robust infrastructure for the integration of MLCA and DFE databases.

2. To facilitate integration of multi-disciplinary, large size data for all MLCA and DFE projects.
3. To provide data integrity and performance for database applications.
4. To provide a common repository of persistent data elements and understand the issues and complexity involved.
5. Provide relations in MLCA and DFE databases such that there will be flow of information among all stage of multi-lifecycle.
6. To import data from CAD packages to MLCA-DFE integrated database programmatically.

1.3Research Need

Design challenges posed by a multi-lifecycle strategy are significantly more complex than traditional product design. The designer must look forward in time to maximize the product's end-of-life yield of assemblies, parts and materials while looking backward to the world of existing products for feedstock sources for the current design. There is a growing body of literature devoted to the development of design for the environment (DFE) tools using lifecycle assessment (LCA) as the mechanism for quantifying the impact of design and production issues over the working life of a product [10].

Consumer pressure, legislation, standards, the need to maintain competitive advantage or the desire to be a good corporate citizen has forced designers and manufacturers to consider the environmental impact of their products. A range of DFE tools is now emerging to assist this process.

As MLCA and DFE share some common data items, such as, part geometry, material and manufacturing process, it is advantageous to integrate the database for

MLCA and DFE. DFE tools must consider not only short-term environmental goals, but also how to achieve the longer-term vision of a sustainable society. In order to achieve sustainable development it is very necessary to reduce the consumption of resources and energy. Computer Aided Design (CAD) tools used for designing the product does not include demanufacturing, remanufacturing or reengineering data that MLCA includes. Thus MLCA takes into account the factors for resources and energy considering the demanufacturing, remanufacturing and reengineering processes. So this integration of CAD and MLCA database will provide not only to designers but also for demanufacturers and MLCA analysts to play an active role in achieving this vision of sustainability.

A longer-term strategy for the design of products or services is necessary to move beyond current incremental environmental improvements. [25] So MLCA-DFE integrated database will contain the data required for calculating environmental burden and also data required for giving intelligent design suggestions such as material selection, manufacturing process or the disassembly effort.

Another problem associated with MLCA software is that the user of MLCA has to provide a significant amount of information about a product for which the environmental burdens are being analyzed. The information required is assembly and subassembly names, parts, materials, manufacturing processes, and the structural relationships between assemblies and its parts (child - parent).

The user has to enter all the information manually, then hundreds or thousands of entries, depending on the assembly and design complexity will need be entered. Entering data is a frustrating and error prone activity for the user. To avoid the manual

work and associative problems, a MLCA-CAD interface has been proposed to programmatically populate the MLCA-DFE integrated database by using the Bill of Material (BOM) information in the CAD software. This MLCA-CAD interface provides a flow of information from design software (DFE/CAD) to MLCA software.

1.4 Multi-lifecycle Engineering

Sustainable development for present and future generations depends on turning environmental responsibility into a competitive advantage through innovative new materials reengineered from waste streams, better products designed for efficient demanufacture as well as manufacture, and agile production technologies that minimizes waste and enhance flexibility, robustness and responsiveness. [11]

To do these, Multi-Lifecycle Engineering (MLCE), a new approach in today's environmental area was proposed [10]. Multi-Lifecycle Engineering is a comprehensive, systems approach to growing a strong industrial economy while maintaining a clean, healthy environment not only today but also for future generations. It is based on the principle of sustainable economy where competitiveness is balanced with environmental responsibility. MLCE takes a systems perspective and fully considers the potential of recovering and reengineering materials and components from one product to create another, not just once, but many times. This is not simply recycling or design for the environment, but rather a complex, next generation engineered system that transcends traditional discipline boundaries in search of scientific knowledge, new methodologies and technologies. The fundamental principles of Multi-Lifecycle Engineering are

- **Resource Conservation and Environmental Responsibility** – Reduce material and energy usage and minimize environmental impact across all stages of the product lifecycle.
- **Asset Recovery** – Mine waste streams and demanufacture products to capture valuable components effectively and efficiently.
- **Materials Reutilization and Life Extension** – Reengineer and remanufacture recovered materials, parts and components efficiently using environmentally conscious robust processes.
- **Multi-lifecycle Thinking and Design** – Design and manage products, processes and packaging with full multi-lifecycle considerations and efficient reutilization of recovered resources.
- **Product Stewardship** – Promote policy, institutional and corporate management infrastructures supporting multi-lifecycle practice and behavior and encouraging future sustainability.
- **Competitive Eco-Products and Innovative Materials** – Explore marketplace opportunities for innovative next-generation applications of multi-lifecycle products and materials where competitive advantage is the bottomline.
- **Integrating Educational Theme** - Create multi-disciplinary and professional programs weaving multi-lifecycle principals into core curricula across the university and beyond.

1.5 Thesis Format

This thesis presents a review of concepts of Lifecycle assessment (LCA), Multi-Lifecycle Assessment, and DFE in Chapter 2. Chapter 2 also focuses on data-flow diagram for MLCA and all stages in MLCA. A review of current database technologies is reported in chapter 3. Chapter 3 gives insight about the types of databases and their comparisons. In chapter 4, the database requirements and different users of the database are described with a focus on the entities and their relationship among each other in the database. An entity-relation schema is mapped to relational and checked for normalization in chapter 4. In chapter 5, MLCA-CAD interface between the CAD software and MLCA-DFE integrated database is described. It includes requirements, the integration flow diagram, input and output specifications of the interface, and implementation details. In Chapter 6, a case study on a *1999 office telephone* is presented to illustrate the integrated database application, including population of the database and relations between each part and its process. Conclusions and further research directions are presented in Chapter 7. Appendix A focuses on the database entity relationship symbols. Appendix B represents source code of file 'Bom.c' which is a part of the code for the interface implementation. Appendix C represents source code of file 'AsmCompVisit.c' which is the other part of code for the interface implementation. Appendix D contains source code for the database population program.

CHAPTER 2

LITERATURE REVIEW

2.1 Design for Environment

Joseph Fiksel defined Design for Environment as *systematic considerations of design performance with respect to environmental health, and safety objectives over the full product life cycle* [16]. Because of this broad definition, the scope of DFE encompasses a variety of disciplines, including environmental risk management, product safety, occupational health and safety, pollution prevention, ecology and resource conservation, accident prevention, and waste management. Figure 2.1 shows a hierarchical breakdown of DFE disciplines.

DFE is "a systematic process by which firms design products and processes in an environmentally conscious way" [16]. It requires environmental considerations over a complete product life cycle in the design process. Closely linking to DFE output, Life Cycle Assessment (LCA), is a family of methods for examining and selecting materials, products, processes, and technologies of a product through every step of its life. DFE considers the disassembly of products at end-of-life, and reveals the associated cost benefits and environmental impacts of a product design. This quantitative information can then be used to make informed trade-off decisions - at the earliest concept stages of design.

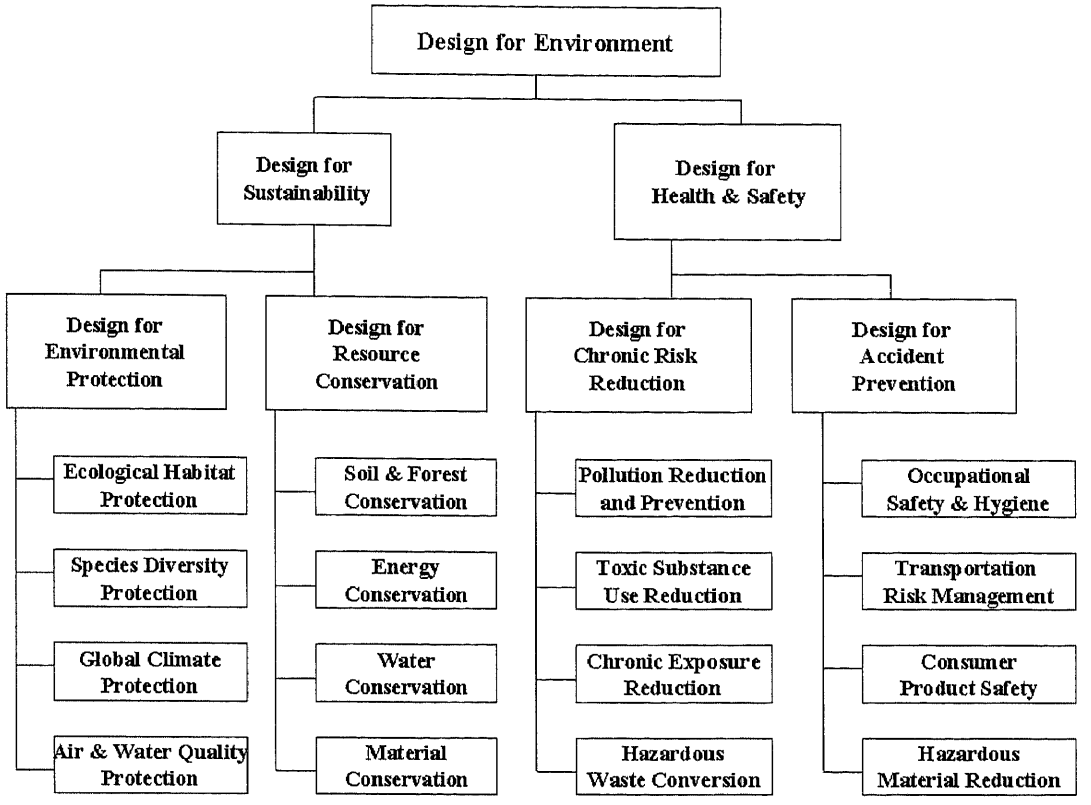


Figure 2.1 A Hierarchy of DFE Disciplines [16]

Tools and methods to assist the integration of environmental considerations into the design process have proliferated in recent years. These tools can assess a product's environmental impact or assist in improving its environmental performance.

DFE tools can be classified into two main groups [25]: analysis tools and improvement tools. These tools vary from the generic analysis to the specific analysis. For example many LCA-based tools are available for general use across a variety of industries whilst improvement tools such as handbooks, databases and checklists tend to be industry specific (and often product specific if developed in-house by companies). Parameters used in analysis of these tools, includes the following attribute:[30]

- Domain - Area in which the tools are used. e.g. tools intended to use for sheet metal machining.
- Stage - Design the most common stage of effort, which can be accomplished by Process design, service, assembly etc. as the stages accounted for, by the tool.
- Knowledge Base – Completeness of resources used for the tool. A list of material with their environmental impacts constitutes a knowledge base.
- Stage Integration – Integration of different stage tools. e.g. design and disassembly, two different stages can be implemented as different tools, although claimed as an integrated modules of a single tool.
- Domain Integration – Integration of different domains. Thus how integrated the tools is to support various aspects of a production with respect to manufacturing process
- Abstraction Level - Incorporate various concerns of environmental policies rather than design level concerns.

2.2 Lifecycle Assessment

Society of Environmental Toxicology and Chemistry (SETAC) has defined Lifecycle Assessment (LCA) as *an objective process to evaluate the environmental burdens associated with a product, process, or activity by identifying and quantifying energy and material usage and environmental releases, to assess the impact of those energy and material uses and release on the environment and to evaluate and implement the opportunities to effect environmental improvement* [15].

LCA is a tool that is targeted at analysis of the design, and involves an examination of all aspects of product design from the preparation of input materials to end-use to evaluate environmental effects of a product, process or activity. This includes analysis of the types and quantities of input materials, water and product outputs, including atmospheric emissions, solid and aqueous waste, and the end product. LCA also incorporates identification and evaluation of different opportunities for reduction of the environmental impacts of processes and products.

LCA methodology has four interrelated components: definition of scope and boundaries, lifecycle inventory (LCI) analysis, impact analysis, and improvement analysis [2]. The goal and scope of the LCA defines purpose of study, boundary conditions and assumptions. LCI quantifies the resource and energy use, and environmental releases associated with the system being evaluated with the system being evaluated. An impact analysis consists of three stages – classification, characterization and valuation. Improvement analysis is a desire to reduce burdens on the environment by altering a product or process [17].

An environmental LCA evaluates the environmental effects associated with any given activity from the initial gathering of raw materials from the earth (petroleum, crops, ores, etc.) to the point at which all materials return to the earth. This evaluation includes all side-stream releases to the air, water, and soil. LCA is an attempt to comprehensively describe all these activities and the resulting environmental releases and impacts.

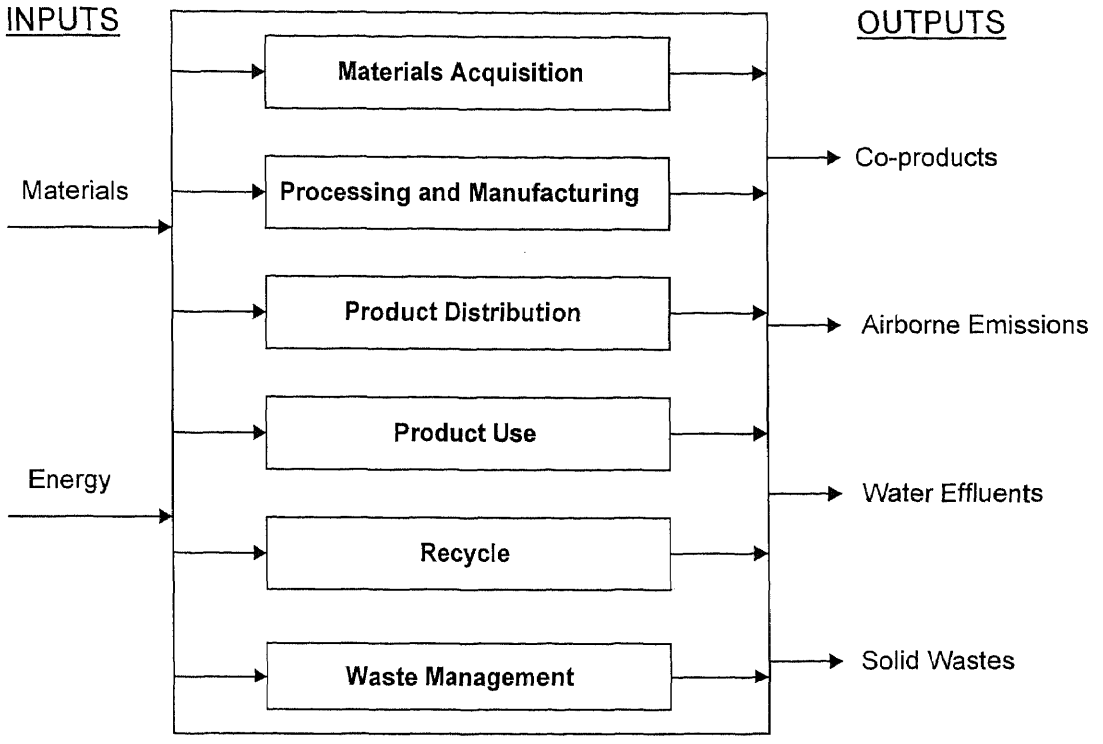


Figure 2.2 Life-Cycle Assessment Stages and Boundaries [23]

LCA requires quantitative and qualitative information that fully describes all life cycle stages to establish the levels and types of energy and materials inputs to an industrial system and the environmental release that result, as shown schematically in figure 2.2 [23]. The approach can be based on the idea of a family of materials budgets, measuring the inputs of energy and resources that are supplied and the resulting products, including those with value and those with potential liabilities. The assessment is performed over the entire life cycle including materials extraction, manufacture, distribution, use and disposal.

2.3 Multi-Lifecycle

Traditional Life-cycle Assessment is a cradle-to-grave analysis whereas multi-lifecycle assessment emphasizes a cradle-to-cradle-to-cradle perspective. Consequently, a gap exists between the current status and future needs for modeling and assessing end-of-life demanufacturing, recovery and reengineering processes necessary for full multi-lifecycle considerations. Multi-Lifecycle Engineering Research Center has extended the structure of traditional LCA set forth by SETAC and EPA, to include explicit consideration of demanufacturing, remanufacturing, reengineering and reuse-extending LCA's to the realm of multi-lifecycle engineering. These end-of-life recovery processes have been modeled to account for material flows, energy usage and environmental burdens associated with recovery and reprocessing of components and basic materials [9].

Traditional lifecycles stage includes 1) Materials production includes two stages, raw materials extraction and materials synthesis. 2) To quantify the materials used in a packaging process, methods of transportation, distance traveled, and energy and emissions associated with these processes, a packaging and distribution stage is separated from the production stage as an independent stage.

MLCA not only includes the stage of traditional lifecycle described above but also includes the stages -1) The recovery and new life options of a product is the main point where MLCA differs from LCA. LCA specifies two types of recycling processes, open loop recycling and closed-loop recycling. MLCA merges these two recycling options into one, and throughout its life from raw material extraction to final disposal. MLCA calls this stage of a product lifecycle as *Demanufacturing*. 2) The *Reengineering* stages acts as a link that closes the material lifecycle loop. 3) *Remanufacturing* stage is

where the parts and subassemblies are reused. Those parts can be reused in new products at production stage or as replacements at the use stage or can be sent back to *demanufacturing*. An important aspect of multi-lifecycle assessment is to balance flows of energy and materials and to quantify emissions, solid wastes and water effluents throughout the product life. It allocates appropriate benefits to the product over multiple generations rather than one.

2.4 Information Infrastructure

2.4.1 Data Flow Diagram

The MLCA model quantifies energy and material inputs and provides the outputs, by-products and environmental impacts of the processes. Through the mass and energy balance equation, the total energy, mass and environmental requirements for the production of each material can be determined. [11] Fig. 2.3 shows the lifecycle data flow in terms of considerations for analysis and modeling.

All the processes are shown in rectangular boxes and the arrows drawn between the processes represents flow between those processes.

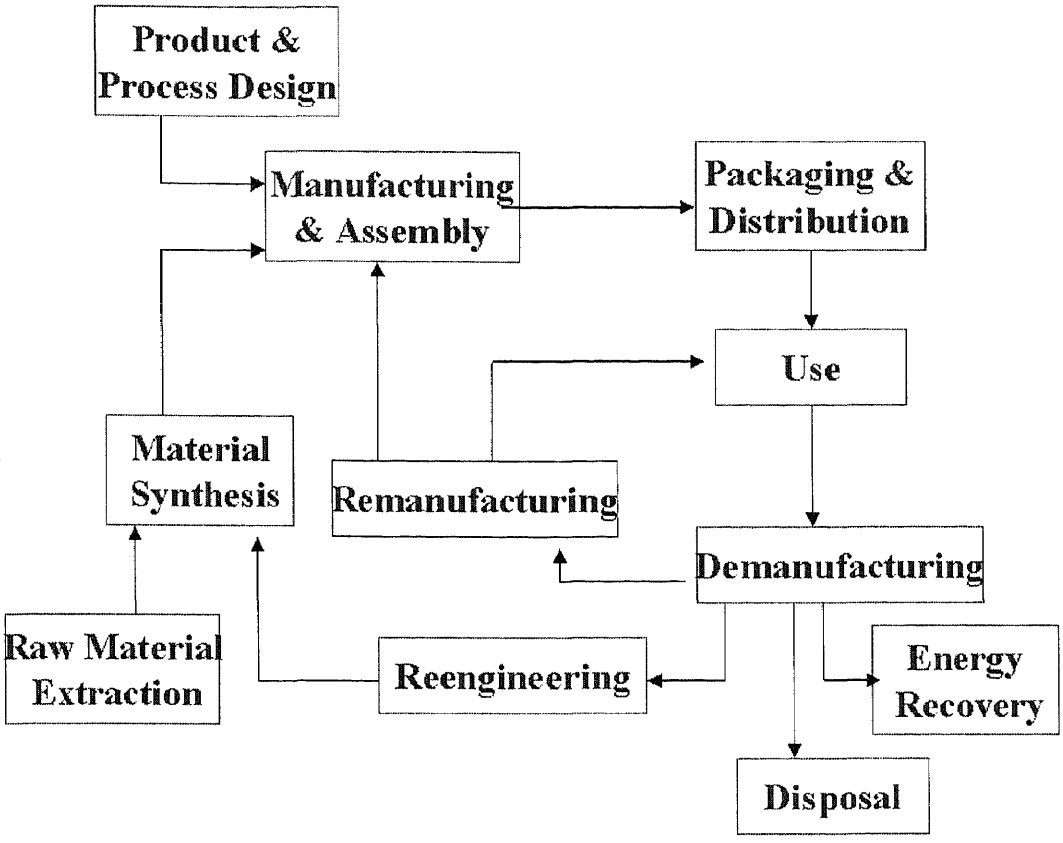


Figure 2.3 Data Flow Diagram on MLCA [17]

2.5 MLCA Stages

As described in section 2.3, MERC has rebuilt the traditional four step lifecycle into seven step multi-lifecycle. These steps are Material Extraction, Material Synthesis, Manufacturing and Assembly, Packaging and Distribution, Demanufaturing, Remanufacturing, and Reengineering. The details of each step are described below.

2.5.1 Material Extraction and Synthesis Stage

From about 90 chemical elements, some of which can not be efficiently recovered, scientist and technologist have been able to refine, combine, and take apart and recombine elements to produce thousands of metals, wood products, plastics and

ceramics that are chemically different and have distinctly different characteristics. Material's shapes and forms have provided the basis from which an almost infinite number of manufactured goods have been developed.

Material production has been divided into two separate stages, materials extraction and materials synthesis. This breakdown is useful in defining the depth and level of the study as it gives more options for designers to evaluate their product based on specific needs. Material Extraction consists of all the processes that are required to extract material from the earth. Materials that can be found and used in their natural form usually require some type of primary processing. For example, sand is washed and graded; trees are cut into boards and planks; and coal is washed, cleaned and sized before it is shipped to the customer. Materials that are combinations of natural materials, such as iron or steel, may require several different processes during the primary stage in order to get them ready for use. Other materials, such as plastics, may be by-products (additional use) of natural materials that are intended for different uses, such as oil for energy. [27]

Material Synthesis consists of intermediate processing, final processing and end-use preparation. Most plastics are formed from natural materials, which are carbon based. They are developed through chemical and mechanical processes. Heat, pressure, and chemical actions are used to separate the atoms and molecules of the raw materials, which are recombined to form new materials. Through the use of mass and energy balance equations, the total energy, mass and environmental requirements for the production of each material were determined.

2.5.2 Manufacturing and Assembly Stage

This stage includes processing of parts, subassemblies and finalsubassemblies. Finalsubassemblies are the assemblies that did not get disassembled during the demanufacturing process. For example in a *computer*, *motor* can be a finalsubassembly that does not get disassembled during the disassembly of the computer. The initial inputs to part production process are reused parts that are recovered in the demanufacturing process and raw/virgin, recycled and reengineered material from other (or same) products. While the inputs to the subassembly production process includes refurbished subassemblies that are also recovered from demanufacturing and feedstock inputs rather than materials. The outputs from these two processes are integrated into the final assembly process, which result in the finished product. [1]

2.5.3 Packaging and Distribution Stage

Another stage that is further quantified in MLCA methodology is the packaging and distribution stage, which has been separated from the production phase as a unique stage. This stage quantifies the materials used in the packaging process, the packaging process itself, as well as methods of transportation, distance traveled and energy and emissions associated with these processes.

2.5.4 Use Stage

Measuring the environmental performance of a product during the use stage is very crucial, since energy consumption during this stage is usually the maximum. Generally, electronic products consume energy during use in four modes: Active, Idle, Power Save,

and Off modes. Energy consumption must be measured in each of these modes, in order to quantify energy consumed during the product lifecycle. Similarly, environmental burdens are measured during those modes and quantified. After the product is consumed and disposed of, it's sent to demanufacturing, where its end fate is identified.[17]

2.5.5 Demanufacturing Stage

If during product design stage, designers did not concern with the disassembly or the reuse of parts, then disassembly tends to be an expensive process and is usually not a viable end of life option. As a result, products having good recycling value are disposed of or recycled in their entirety. For example in the case of computers and copiers, parts such as CRT glass, keyboard caps, drive motors, transformers, and steel casing, could all be given a new life if they could be disassemble efficiently. [12]

Disassembly is the process of physically separating parts and subassemblies in a product. To recover the maximum value from the discarded products, product disassembly stage needs to be carefully analyzed in terms of sequence dependency between components and the required level of disassembly.[13]

The process of disassembly involves several activities, including collection of parts, sorting of parts, product/part handling, disassembly instruction, part separation, part cleaning, part testing and inspection.

These activities can further classified into 1. Unfastening Processes – where a fastening devise is removed in an operation, which reverses the assembly fastening action. 2. Disassembly Processes – All other activities that facilitate the separation of the product into its parts. [13]

2.5.6 Reengineering Stage

The main option to consider and where MLCA differs from LCA is in the recovery of material and new life options of the product. LCA addresses two types of recycling processes, open loop recycling and closed loop recycling. In open-loop recycling system, a product made from virgin material is recycled into another product that is not recycled, but disposed off, possibly after a long-term diversion, thus giving only one life to the product. Whereas closed-loop recycling occurs when a product is recycled into a product that can be recycled over and over again. LCA looks into this as two separate distinct recycling options. This is one area where MLCA plays its role, in addressing these two recycling options simultaneously, rather than in isolation, at the end of product's life as well as throughout its life. [17]

MLCA tries not to landfill any material as far as possible. It finds new options for the waste disposed at every stage so that it can be reengineered into useful products and not just once but again and again. This reengineering stage acts as a link that closes the lifecycle loop. Reengineering involves characterization of waste streams and the reformulation of materials derived from waste streams. Five major reengineering processes were identified for recovered materials: Reprocess, Compatibilize, Pyrolysis to Fuels, Pyrolysis/Hydrolysis to monomers, and shredding of metals. The major material inputs to this stage are obtained from demanufactured products, through process such as shredding and separation [28].

2.5.7 Remanufacturing Stage

The remanufacturing stage is where the parts and subassemblies are refurbished. These refurbished products could then be used in new products at the production stage or for replacements and maintenance at the use stage or could be sent for demanufacturing. Parts and subassemblies entering the remanufacturing process can either be from demanufactured products, or from the manufacturing process. The flow of remanufacturing process is as follows:

Sort → Pretest → Disassemble → Clean → Rebulid/
Upgrade → Reassemble → Final Test → Packaging → Transport

Energy and environmental burdens must be quantified at each of the above steps. The result is either an assembly or a product that is fed back into the manufacturing process or back to the use stage through maintenance. Also, products and subassemblies that can not be remanufactured are sent to demanufacturing for further processing [1].

CHAPTER 3

DATABASE CONCEPTS

3.1 Database Management System

Database Management Systems (DBMS) takes a central role in applications where large persistent collection of data is to be organized and maintained. Hence DBMS is supported by a computer based information system. A database system consists of software, the database management system and one or several databases. Database represents some aspect of real world, called as *miniworld* or *Universe of Closure*. Thus a database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database. A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested.

A database management system (DBMS) is a general-purpose software system that facilitates the process of defining, constructing, and updating database for various applications. Defining a database involves specifying the data types, structures, and constraints for the data to be stored in the database. Constructing the database is the process of storing the data itself on some storage medium that is controlled by the DBMS. Manipulating a database includes functions such as querying the database to retrieve specific data, updating the database to reflect changes in the miniworld, and generating reports from the data [14].

3.1.1 Database Management Systems Architecture

The important characteristics of a DBMS design approach are a) Insulation of programs and data. b) Support to multiple user views C) Use of a catalog to store database description (schema). The most common architecture is called a three-schema architecture [14].

The goal of this three-schema architecture, illustrated in figure 2.1, is to separate the user applications and the physical database, thus making them independent. In this architecture, schemas can be defined at the following levels:

1. The internal level has an internal schema, which described the physical storage structure of database. The internal schema uses a physical data model and describes the complete detail data storage and access paths for the database.

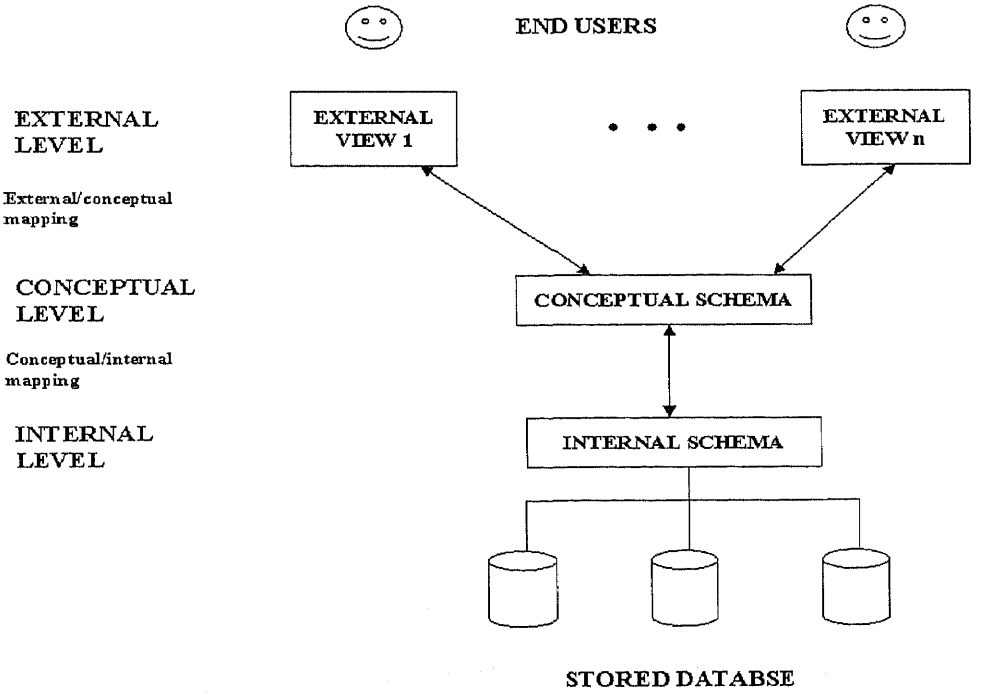


Figure 3.1 The Three Schema Architecture [14]

2. The conceptual level has a conceptual schema, which describes the structure of the entire database for a community of users. The conceptual schema hides the details of physical storage structures and concentrates on describing entities, data type, relationships, user operations and constraints. A high-level data model or an implementation data model can be used at this level.
3. The external or view level includes a number of external schemas or user views. Each external schema describes the part of the database that a particular user group is interested in and hides the rest of the database from that user group. A high-level data model or an implementation data model can be used at this level.

3.2 Classification of Database Management Systems

A data model is a set of concepts that can be used to describe the structure of the database. The common criterion used to classify DBMS is the data model. The data models most often in commercial DBMS are Relational and Object-Oriented models. The details about the models are described in the following sections.

3.2.1 Relational Theory

The relational systems are based on a simpler approach to organize the data in relations or tables, which allows a considerably clearer distinction between a logical and physical data model. The theory of relations is simple and elegant. The frameworks of relational algebra and relational calculus are based upon fundamental mathematical notions like sets, functions and predicate logic. Hence relational languages can be non-procedural and declarative like SQL. Relational system offers a high degree of physical data independence [18].

Properties of relational databases must be expressed in terms of attributes names and relation instances. Examples of such properties are functional dependencies between the attributes of a relation schema or inclusion, transitive dependencies between the attributes and join dependencies relating instances of different relational schema. Based on these dependencies and relations, different grouping of attributes can be done. To compare these different grouping or different database designs, the definition of the well-known framework of normal forms is made. Normalization is a measure of the appropriateness or quality of design, other than the intuition of designer.

In the relational database, the dynamic of a database is not handled explicitly. Mostly dependencies are used as input for normalization algorithms to control redundancy of data and to avoid the update anomalies. Normalization algorithms are formally specified and realized in relational database design tools. In relational theory, there are tools for analyzing redundancy in storing data. However, this well-developed normalization theory brings with it the danger of ignoring other design requirements that are not part of normalization. For example, if focus is made on structural redundancy only, there are chances to forget about other constraints and database properties. The framework for relations helps to decide redundancy properties on the one hand, but hinders the adequate modeling of other data structures like lists and arrays (which are usually connected with the use of specific operations sometimes not even expressible in languages like relational algebra). Such data structures have to be modeled in an artificial way, and relational design theory will ignore the special properties of them [26].

Dynamics in relational database theory is purely based on the generic operations - *Insert, Delete and Update*. Database items are characterized by their structure, specific functions applied to them and the temporal evolution of the stored data.

3.2.2 Object Oriented Theory

Object-oriented data models provide a semantically richer framework for modeling and implementing complex information systems. In the last ten years several object models that support many different concepts were introduced in order to fulfill the demands of advanced applications.

The term (object) *type* has an intentional character and refers to definition of common properties (structure and behavior) of a set of similar objects, whereas the term *class* describes the extensional character of a set of similar objects. Classes and types are usually organized in inheritance hierarchies. In case of type hierarchy, the inheritance mechanism is used to factorize shared specifications and implementations. A subtype of an object type specializes the properties of that object type by redefining existing properties and adding new properties, e.g. a *student* may have additional attributes like university or faculty beside the inherited ones of a *person*. In case of class hierarchy, objects are grouped into several extensions. Here, a subclass specializes the extension of a super-class, the objects of the subclass represent a subset of the objects of the super-class, e.g. each student is a person [22].

In general, classifying, typing and using inheritance helps to better structure and describe a database schemas. Besides, (syntactical) inheritance produces compact code by reusing existing attributes and methods of an object type, and thus reduces the costs for the programming. A complex object (or composite object) may be composed from

several objects that may be composite themselves, e.g. a car is built from (among others) a chassis, four wheels, and a motor which itself is a complex object with many components.

Following the idea of abstract data types (class), objects are separated in an interface and an implementation part. The interface part is the specification of a set of property by defining methods that may be used to access and modify the attribute values of an object. Thus, the interface captures the external view to the applicants of the database on an object. The implementation part, on the other hand, contains the details of an object (data as well as method implementations) [14].

By encapsulating the internal representation of an object, logical data independence is achieved. Changes of the internal structure and/or implementation do not require the modification of applications that are using that object type. Thus, encapsulation helps to maintain complex software by providing a means for structuring systems in independent, easily changeable modules. Moreover, encapsulation can also be used for authorization issues, e.g. integrity or access control conditions can be checked in the methods provided by the interface [18].

3.3 Comparison of Object Oriented and Relational Database Theory

When modeling real world entities in terms of relations, the behavior of the entities is always represented in application programs. In contrast, the object-oriented paradigm integrates structure and behavior of real world entities into coherent units, the so-called *objects*. The Structure of an object is represented by a set of attributes while the behavior is expressed by a set of operations (usually called methods) that are applicable for that object. By integrating structure as well as behavior of real world entities, object-oriented

database design succeeds in capturing more semantic of applications already in the design phase.

For building composite objects, a set of type constructors like tuple-of or set-of are provided by object-oriented database systems. In general, these constructors are applicable in a orthogonal fashion. In comparison with that, the constructors of the relational model do not meet this requirement, because the only existing abstraction is the relation, which is a set of tuples of atomic values. Furthermore in opposition to relational database systems, object-oriented database systems allow the transitive propagation of an operation on a composite object to all its components [22].

The concept of composite objects is an important means for more realistic modeling. Especially in case of modeling relationships and dependencies between objects, the benefits of this concept are obvious. In relational databases, tuples are identified in a relation by a given key, whereas in object oriented databases, objects are uniquely distinguished and accessed by object identifiers, which are independent from their attribute values.

The object identity approach has several advantages. Objects can refer to other objects simply by using their object identifier. A schema designer, therefore, is not responsible for finding adequate keys for various classes of objects in order to express referential relationships. Obviously, there is no need to introduce artificial attributes, for instance, an employee or social security number in a person class.

In the object-oriented area, there exists a number of design approaches. Those approaches often claim to cover the whole spectrum from requirement analysis to system design (and sometimes even up to the implementation). In addition, they are often

proposed to be ideal for object-oriented database design. However, these object-oriented methodologies cannot directly be used for designing object-oriented databases because they neglect or even ignore certain properties expected in the context of databases (integrity constraints, transactions, etc.). Another object-oriented approach to conceptually modeling of information systems is a formal specification approach, which is based on a temporal logic framework for objects. In contrast to the object-oriented methods described above, relational database has a clear and sound formal semantics. At the time being, all these object-oriented approaches fail in providing a similar, well-developed theory like that of relational database design. For instance, there are no quality criteria available, which could be used for some kind of object-oriented normalization of database designs in order to achieve the best possible design [22].

The hybrid database (or sometime called enhanced E-R) carries advantages of relational and object oriented database. Hence for MLCA and DFE integration a hybrid database is chosen. The basic entities and relations are represented in E-R diagram while the new entities, which inherit properties from the basic entities are represented as subclass.

3.4 Database Design Process

A database structure's correctness is measured by how well it can serve the purpose to corresponds with the real world concepts it is supposed to be represent. Hence a database structure's correctness is a critical factor in the task of designing the database.

The first step in designing the database is to identify and understand the real world concepts of the actual problem domain. This step requires a conceptual model to be established which represents the concept of the problem domain at a higher level of

abstraction. Such a model allows for concentrating on general topics and principles without having to engage in technical matters more related to database technology. This conceptual model referred as data model, deals with the definition of entities and their relations. Some entities are concrete in nature and are therefore easily identified, but others can be far more abstract and sometimes harder to identify. *Component, Material* are examples of the former and an event such as *Manufacturing Process* can serve as an example of latter.

A data model is strongly formalized, and is easily transformed into a database structure. This transformation is quite straightforward, and most of the entities of the model will have a corresponding table in the database.

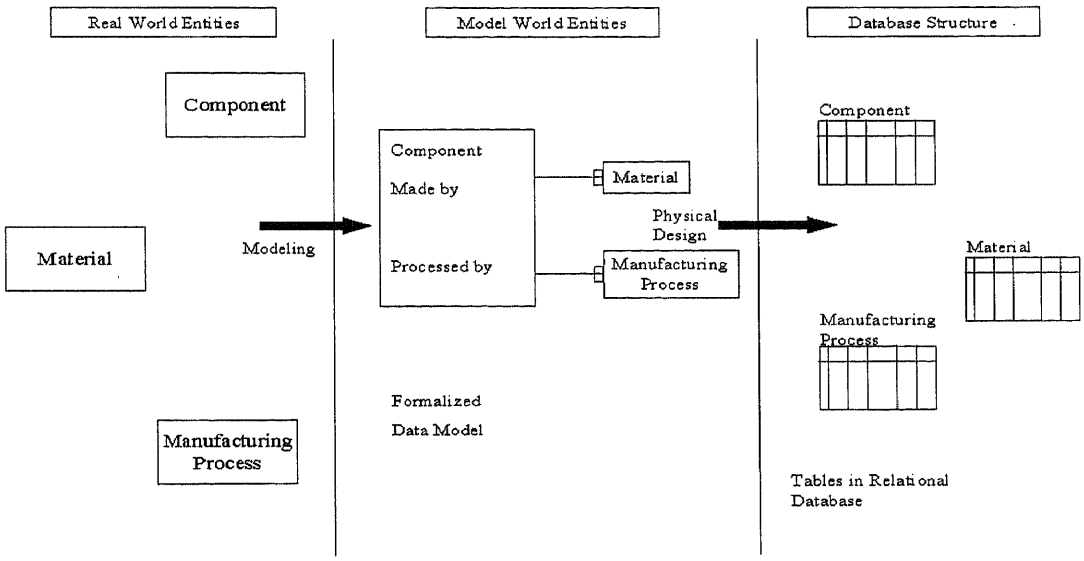


Figure 3.2 The Database Design Process [22].

The establishment of a data model is an important part of design process but it also serves as a way of understanding the database structure and how it can be exploited

into different applications. Accordingly the description of database is divided into two parts where the first one mainly addresses the data model and the second one addresses the actual database structure. (tables and their contents)

3.4.1 Data Model Terminology

Data Models are based on three main components.

- Entities – which are model representations of real world concepts
- Attributes – by which entity is described. For example an entity *Component* is described by the attributes name, color, and weight.
- Relations – shows how the different entities are associated with each other. For example *Component* and *Material* can be associated to each other by the relation *Made by*.

3.5 Public Databases for MLCA

The conduct of MLCA is a data-intensive endeavor. The database for LCA can be private or public but it increases the efficiency of performing as LCA of a product. The databases can be divided into three types : [31]

- Non-bibliographic database containing on-line information covering resource use; energy consumption; environmental emission; and chemical, biological or toxicological effects.
- Database clearinghouses, which are online services, both government-run and private, that facilitate the retrieval of information from a variety of databases and bibliographic.

- Bibliographic database which are on-line database containing bibliographic references to the actual data which must be extracted from the referenced sources.

External databases are those that are available as distinct information products separate from LCA software systems. Although a significant fraction of LCA data comes from product, material production (cradle-to-gate), or study-specific data collection efforts, the reminder is derived from so-called secondary data sources. Secondary data are defined as publicly available data that have not been collected specifically for the purpose of conducting LCA and for which practitioner has no input into the data collection process.

Secondary data source include [31] -

- Reference books, such as the Encyclopedia of Chemical Technology
- Industry reports, such as SRI International studies on economic and production statistics
- Environmental Protection Agency databases covering chemical emissions and dispositions
- Department of Energy database, especially those of the Energy Information Administration
- Department of the Interior data on Natural Resources

Very few LCA databases are public. The integrated database of MLCA and DFE should have the capability of sharing the information with the public database.

CHAPTER 4

DATABASE DESIGN

4.1 Database Requirements

The three main users for MLCA and DFE integrated database are Product/Process Designer, Demanufacturer, and MLCA Analyst. All users have some common tasks and some are specialized to their roles. The details of each role are given below.

4.1.1 Product/Process Designer

The designer must look forward in time to maximize the product's end-of-life yield of assemblies, parts and materials while looking backward to the world of existing products for feedstock sources for the current design. Designer has to use design for the environment (DFE) tools using lifecycle assessment (LCA) as the mechanism for quantifying the impact of design and production issues over the working life of a product. To satisfy these requirements, the database should contain the following information

a) Tree structure of all the components in a product.

b) Complete description of part, subassembly and final subassembly.

c) part's material, weight, quantity, and manufacturing process(es).

d) Feedback from the demanufacturer about the demanufacturability of the product.

4.1.2 Demanufacturer

Demanufacturing is a main decision process about products remanufacturability or reuse giving multiple life to the product. In this stage, the decision about reengineering,

remanufacturing and reuse is taken. Demanufacturer is interested in knowing the process plan of demanufacturing, and optimization of that plan to get maximum benefit.

To satisfy these requirements, there should be flow of information from the design stage to demanufacturing stage in the database. Along with that, the database should contain the following information

a) Information about demanufacturing facilities.

b) The connectivity between two parts and also how they are connected.

c) The time required to disassemble the parts.

d) The tools needed to perform the disassembly.

4.1.3 MLCA Analyst

The analyst will be interested in all the stages of a product and will postulate and analyze the "What-if" scenarios. The analyst will determine the environmental performance of the product and identify changes in that product or process to improve its environmental footprint while being economically better and higher quality.

To satisfy these requirements and performing the analysis information from the design stage to the demanufacturing stage should flow into the database. Thus the database should contain the following information : -

a) Material extraction and synthesis stage details.

b) Manufacturing process details.

c) Packaging and transportation details.

d) Use stage details.

e) Remanufacturing details.

f) Reengineering stage details.

The details in each stage include the information about environmental impacts, embodied energy and cost.

4.2 Entity Description

An entity is defined as "the thing that can be distinctly identified " [29]. A group of similar entities form an entity set. Entities have properties, called attributes, which associate a value from the domain of values for that attribute with each entity in an entity set. The selection of relevant attributes for the entity set is a crucial step in the design of a real world model. An attribute or a set of attributes whose values uniquely identify each entity in an entity set is called a key for that entity.

For the MLCA and DFE integrated database, the main entities are component, material, process, and facility. MLCA key factor is a weak entity as its existence is fully depend upon on the process entity.

4.2.1 Component

As a product level description, the database is required to store information about product composition i.e., the parts, subassemblies and final-subassemblies. The entity type COMPONENT has additional subgroupings of its entities like PRODUCT, SUBASSEMBLY, FINALSUBASSEMBLY and PART that are meaningful and need to be represented explicitly because of their significance to the database applications. Hence COMPONENT can be a super-class and PRODUCT, PART, SUBASSEMBLY and FINALSUBASSEMBLY can be formed sub-classes. The subclass will inherit all the super-class property. This subclass-superclass relationship is illustrated in figure 4.1.

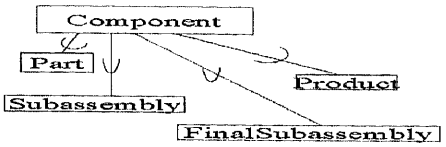


Figure 4.1 A superclass entity COMPONENT with its subclass PART, SUBASSEMBLY, FINAL-SUBASSEMBLY and PRODUCT

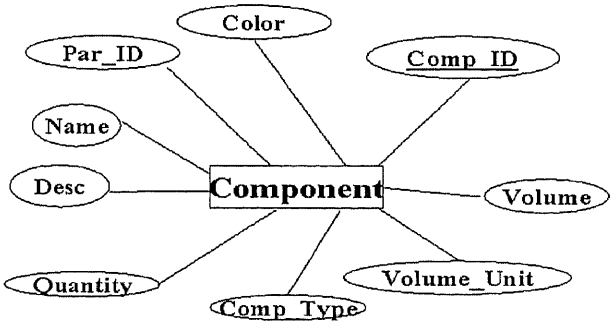


Figure 4.2 An entity COMPONENT with its attributes

For entity type component, the attributes are a unique component identity number viz. *Comp_ID*, *name, description* (functions), *color, volume and quantity, comp_type*. The entity Component with its attributes is shown in figure 4.2. The attribute *Comp_Type*

differentiates between the PART, SUBASSEMBLY, FINAL-SUBASSEMBLY and PRODUCT.

A PRODUCT is composed of final-subassemblies, subassemblies and parts. A SUBASSEMBLY itself consists of other subassemblies or individual parts. Hence, to represent this recursive relation between subassemblies and parts, an attribute *parent_ID* is formed. Product would have *parent_ID* as zero as it does not have any parent. The PART which does not have its *Comp_id* as *parent_ID* is the final part without child. Thus by tracking the Comp-ID and Parent ID, a tree structure can be traced.

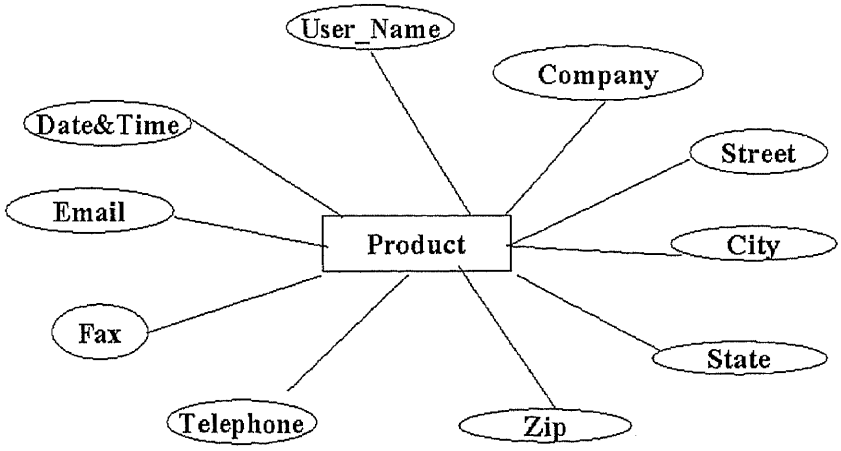


Figure 4.3 An entity PRODUCT with its attributes

For a PRODUCT, the specialized attributes are *user_name, company, address (*consists of street, city, zip and state*), telephone, fax, email* and *date and time*. Thus

PRODUCT class will have all the attributes of super-class and also all above mentioned specialized attributes. The entity PRODUCT with its specialized attribute is shown in figure 4.3.

4.2.2 Material

For MLCA, the main aspect is material flow through all stages of life of a product. The entity MATERIAL has attributes such as, a unique material identity number viz. *Material_ID, name, market price, aliases, composition name* and *percentage, density, yield rate, reengineered content, recycled content* and *virgin content*. As a single material can consists of many composites, the attributes *composition name* and *percentage* are multi-valued attribute; hence they are represented in double oval. The entity MATERIAL with its specialized attributes is shown in figure 4.4.

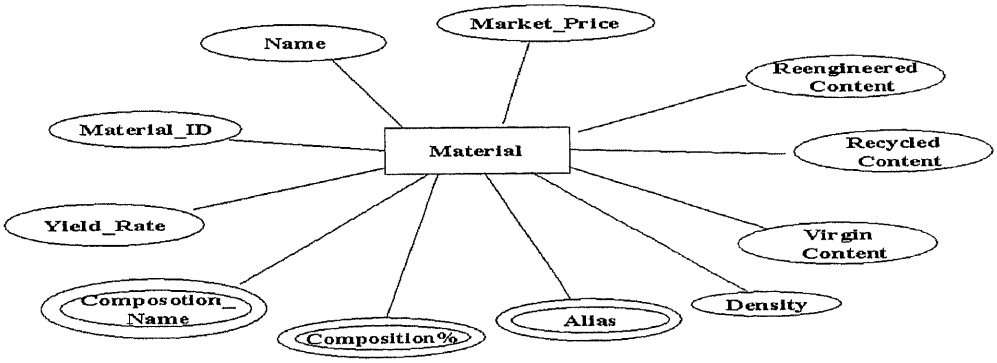


Figure 4.4 A entity MATERIAL with its attributes

4.2.3 Facility

To perform any kind of process, facilities are required. The facilities considered in this thesis are demanufacturing, manufacturing and transportation facilities. To represent any kind of facility, a super-class named FACILITY is formed. This FACILITY class has attributes that are common to all facility and thus this class is formed by the generalization method. The attributes of class facility are *Facility ID, name, description, size of facility, annual equipment cost, annual energy consumption* and *cost per hour*. The entity FACILITY with its attributes is shown in figure 4.5.

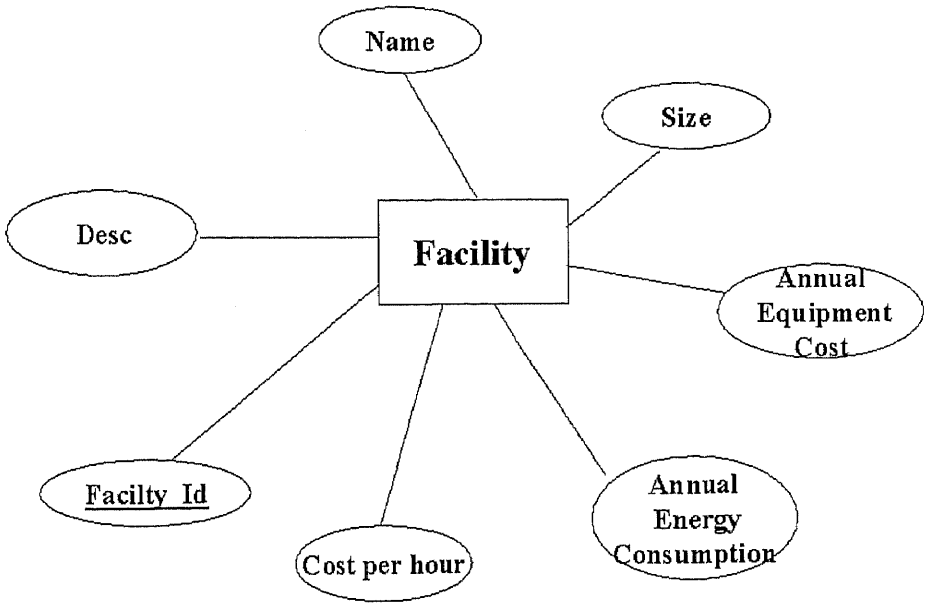


Figure 4.5 A entity FACILITY with its attributes

The subclass and super-class relationship is shown in figure 4.6. The subclasses for Facility class are DEMANUFACTURING FACILITY, DISTRIBUTION FACILITY and MANUFACTURING FACILITY. The specialized attributes for subclass DEMANUFACTURING FACILITY are *volume handled, disassembly time, energy consumption* and *energy cost* for conducting a process. Similarly for subclass DISTRIBUTION FACILITY, the specialized attributes are type of *vehicle used, description of that vehicle* and *load to consumption ratio* for that vehicle.

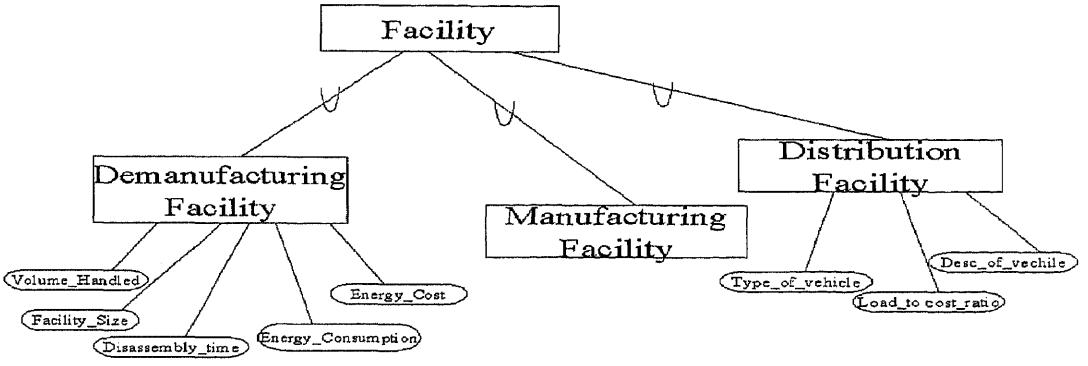
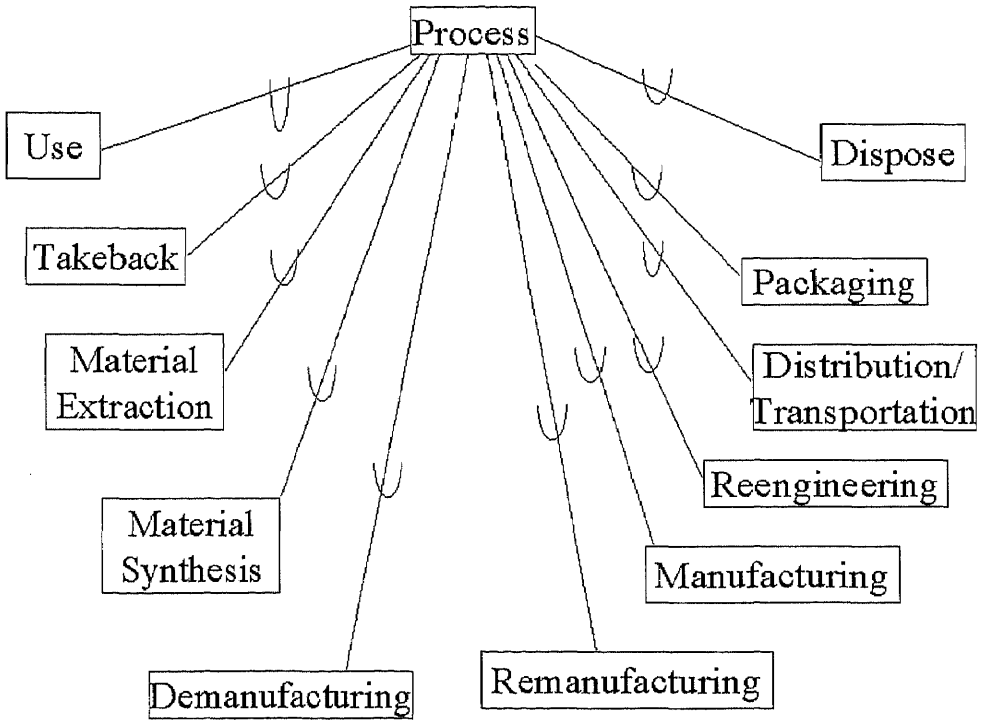


Figure 4.6 A entity FACILITY with its subclasses as DEMANUFACTURING, MANUFACTURING and DISTRIBUTION FACILITY

4.2.4 Process

All MLCA stages are processes or systems of processes. As described in chapter 2, from a material perspective, the main two processes are material extraction and material synthesis. From product and material together, the key processes are Manufacturing and Assembly, Packaging, Use, Demanufacturing, Remanufacturing, and Reengineering.

By the generalization method, gathering all the common properties together, a super-class called PROCESS is formed. All the processes, which are stages of MLCA, are made subclass. The attributes of super-class process are a unique process identity viz, *Process_ID, name, type, yield rate, process time, labor time* and *labor rate* for that particular process. The attributes which are not part of super-class, and which are related to a particular process, became the attributes for a subclass. The subclass and superclass relationship for the entity PROCESS is shown in figure 4.7.



11

Figure 4.7 A entity PROCESS with its subclasses as USE, TAKEBACK, MATERIAL EXTRACTION AND SYNTHESIS, DEMANUFACTURING, REMANUFACTURING, MANUFACTURING, REENGINEERING, DISTRIBUTION, PACKAGING, and DISPOSE

For USE stage the attributes are the attributes which get inherited from super-class and also are energy in power save mode, active mode and idle mode as well as life span.

For MATERIAL EXTRACTION and SYNTHESIS, the main specialized attributes are *step process* (process which are part of material extraction and synthesis) and the *binders* (or catalyst) for that process.

For PACKAGING subclass, the specialized attributes are *packaging material, % recycled, total weight of packaging,* and *volume packaging per product.* For DISTRIBUTION (Transport), the attributes are *distance traveled, actual load,* and *load ratio.*

For DEMANUFACTURING subclass, the specialized attributes are *reselling price, actual rate* and *pass rate.* For the subclasses REMANUFACTURING and REENGINEERING, the study of attributes and their values is going on. They will be added to database as get discovered.

4.2.5 MLCA Key Factors

The outcome of each process can be referred by environmental burdens produced and energy and cost associated with that process, which are also important factors for conducting MLCA. Thus to represent this outcome, an entity named MLCA KEY FACTOR is formed. This is a weak entity as its existence depends upon process. The super-class MLCA KEY FACTOR is formed by generalization method and POLLUTION INGREDIENTS, ENERGY and COST are formed as subclass.

The environmental impact is measured in terms of *WE_Al, WE_Ni, WE_Iron, WE_Hg, WE_Oils, WE_Phenols, WE_Lead, WE_Cynaide, WE_ammonia, WE_Sulphides, WE_COD, WE_BOD, WE_Acid, WE_chromium, WE_Metal_Ions, WE_suspended_Solids, WE_Dissolved_Solides, Solid_Wastes, AE_Aldehydes, AE_BOG, AE_CO2, AE_CO, AE_SO2, AE_Monoxide, AE_Ammonia, AE_Nox, AE_Particulate, AE_other_org., AE_Methane, AE_Hydrocarbons, AE_Hg_Vapour, AE_Gas_Dust, AE_Flouride, AE_Cl.* (Note- In all the above attributes, WE means water emission and AE means Air Emission) Hence all factors mentioned above become the attribute for the entity POLLUTION INGREDIENTS. The entity POLLUTION INGREDIENTS with attributes is shown in figure 4.8.

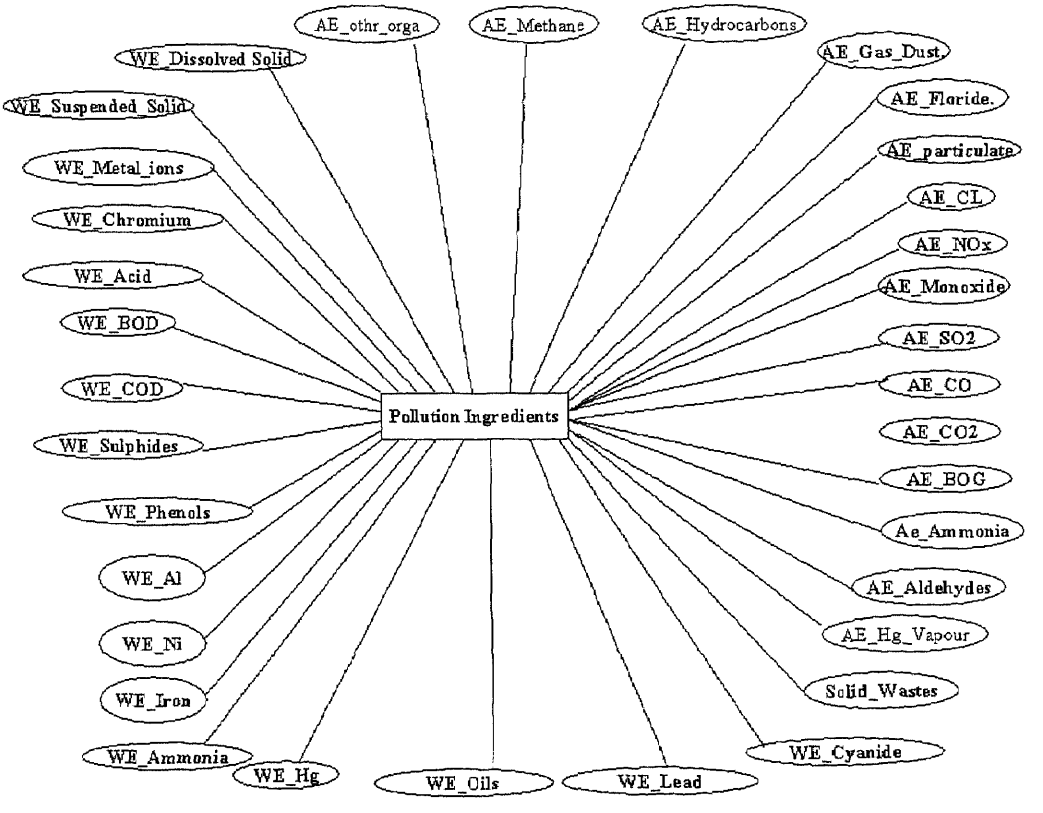


Figure 4.8 A entity POLLUTION INGREDIENTS with all its attributes

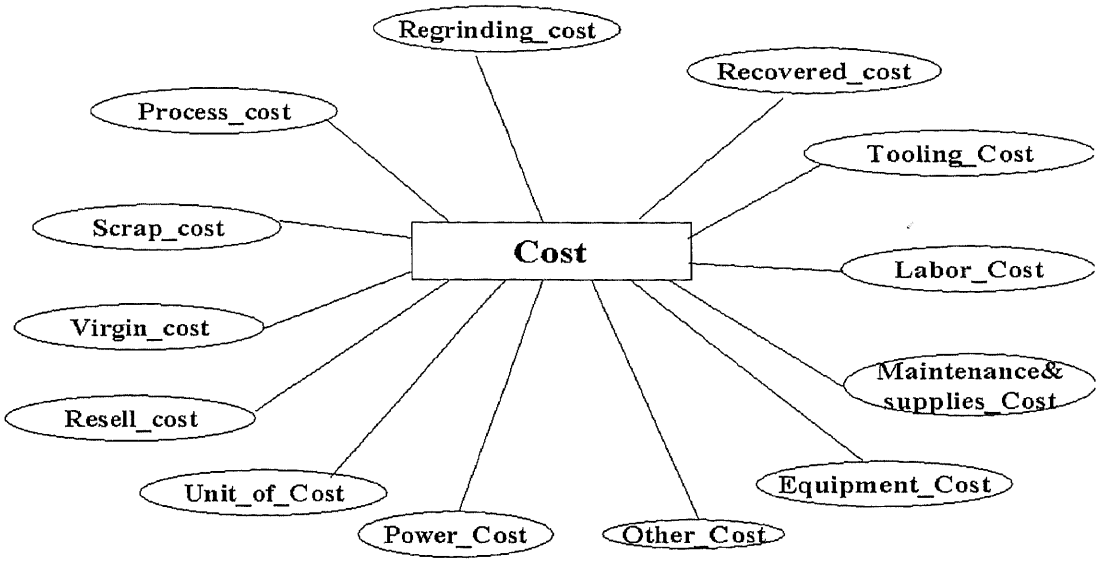


Figure 4.9 An entity Cost with its attributes

Cost can be classified as positive and negative cost depending upon whether the economic exchange is a gain and loss in the process according to MLCA. For this subclass COST, the attributes are *process cost, regrinding cost, recovered cost, tooling cost, labor cost, maintenance & supplies cost, scrap cost, virgin cost, equipment cost, resell cost, power cost, other cost* and *unit of cost*. The entity COST with its attributes is shown in figure 4.9.

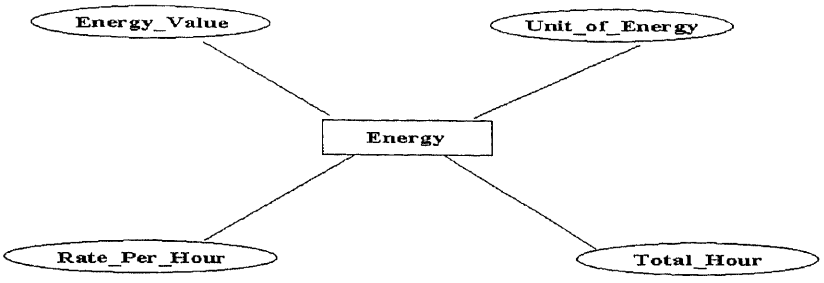


Figure 4.10 An Entity Energy with its Attributes

The subclass energy represents the energy consumption for all processes. The specialized attributes for this class are energy value and its unit, rate per hour and total hour energy/power supply is used. The entity Energy with its attributes is shown in figure 4.10.

4.3 Data Modeling

The data modeling process includes types, entities, relationships, and properties, with defining attributes such as uniqueness, rules, and cardinality. The universe of modeling techniques and notations includes data flow diagrams (DFD), entity-relationship diagrams (ERD), and others techniques. [30] The data flow diagram of MLCA is described in chapter 2 in section 2.4. The relationships between the entities described in above section 4.2 are stated below.

4.3.1 Relationships

A relationship type R among n entity types E_1, E_2 .... E_n defines a set of associations among entities from this type. Mathematically, R is a set of relationship instances r_i, where each r_i associates n entities (r_1, e_2... e_n), and each e_j in r_i is a member of entity type E_j, 1<=j<=n. Informally, each relationship instance r_i in R is an association of entities, where the association includes exactly one entity from each participating entity type.

Each entity type that participates in a relationship type plays a particular role in the relationship. The relationship name signifies the role that a participating entity plays in the relationship instance. Relationship types usually have certain constraints that limit the possible combinations of entities that may participate in relationship instances. These constraints are determined from the miniworld situation that the relationships represent. The cardinality ratio specifies the number of relationship instances in which an entity can participate. For the integrated database, the following relationships are specified.

4.3.1.1 Made by: This relationship represents the connection between the entities PART and MATERIAL and also between FINALSUBASSEMBLY and MATERIAL. Every PART and FINALSUBASSEMBLY can be composed of at least one MATERIAL and upto a maximum of 'n'. Similarly a MATERIAL can be used in zero component or upto a maximum of 'n' components. Hence cardinality ratio is 1:n. This relationship has attribute *weight of material* used for that PART.

4.3.1.2 Processed by: This relationship represents the connection between PART/FINALSUBASSEMBLY and PROCESS and also between the MATERIAL AND PROCESS. Every PART/FINALSUBASSEMBLY can undergo minimum one

PROCESS and upto a maximum of 'n'. And even each PROCESS can be used by minimum zero PART/FINALSUBASSEMBLY to maximum 'n' PART/FINALSUBASSEMBLY. Similarly each MATERIAL can undergo maximum 'n' processes and each PROCESS can use upto a maximum of 'n' MATERIAL.

4.3.1.3 Used by: Every facility can be used by specific process and upto a maximum of 'n' processes. This *Used by* relation is represents the connection between PRODUCTION PROCESS and PRODUCTION FACILITY, PACKAGING PROCESS and PACKAGING FACILITY, DEMANUFACTURING PROCESS and DEMANUFACTURING FACILITY, REMANUFACTURING PROCESS and REMANUFACTURING FACILITY and also BETWEEN REENGINEERING PROCESS and REENGINEERING FACILITY. This relationship has attribute time that the facility is used for that process.

4.3.1.4 Produces: This relationship represents the connection between PROCESS and MLCA KEY FACTOR. It is 1:1 relationship, as each PROCESS produces MLCA KEY FACTORS. Also each MLCA KEY FACTORS is associated with one PROCESS.

4.3.2 Entity Relational Diagram

In Entity-Relationship diagrams, the emphasis is on representing schema rather than the instances. This is more useful because a database schema changes rarely, whereas the extension changes frequently.

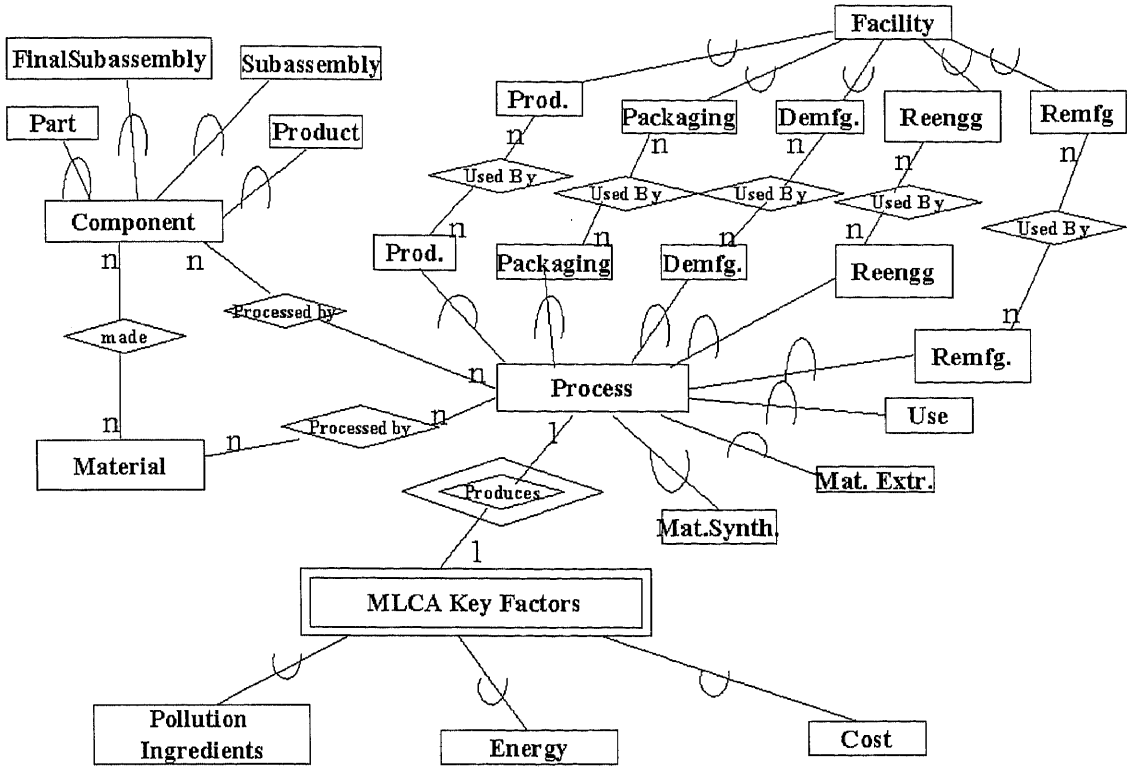


Figure 4.11 Entity Relationship Diagram for MLCA and DFE integrated Database

Entity types such as COMPONENT, PROCESS, MATERIAL, FACILITY are shown in rectangular boxes. Relationship types such as *Processed by, Used, Produces* are shown in diamond-shaped boxes attached to the participating entity types with straight lines. Attributes are shown in ovals, and each attribute is attached to its entity type or relationship type by a straight line. Multivalued attributes are shown in double ovals. Weak entities are distinguished by being placed in double rectangles. Because of space constraints, all attributes of all entities are not shown.

4.3.3 ER-to-Relational Mapping Algorithm

Many Computer Aided Software Engineering (CASE) tools are based on the ER model and its variations. These computerized tools are used interactively by database designer to develop an ER schema for database application. Many tools use ER diagrams or variations to develop schema graphically, and then automatically convert it into a relational database schema in the Data Definition Language (DDL) of a specific relational DBMS.

For MLCA and DFE integrated database, an ER-to-Relational Mapping Algorithm is used [14]. The algorithm is as follows :

Step –1: For each regular entity type E in the ER schema, create a relation R that includes all the simple attributes of E. Include only the simple component attributes of a composite key. Choose one of the key attributes of E as primary key for R. If the chosen key of E is composite, the set of simple attributes that form it will together form the primary key.

For the MLCA integrated database, the relations COMPONENT, MATRIAL, FACILTY, and PROCESS are created. The primary key for the entity COMPONENT, MATERIAL, FACILITY, and PROCESS *are Comp_ID, Material_ID, Facility_ID, Process_ID* respectively.

Step-2: For each weak entity type W in the ER schema with owner entity type E, create a relation R, and include all simple attributes of W as attributes of R. In addition, include as foreign key attributes of R the primary key attribute(s) of the relation(s) that correspond to the owner entity type(s); this takes care of the identifying relationship type of W.

For MLCA integrated database, there is only one weak entity MLCA KEY FACTOR. This entity depends upon on the entity PROCESS. The primary key of entity PROCESS, *Process_ID* becomes the foreign key of entity MLCA KEY FACTOR

Step-3: For each binary 1:1 relationship type R in the ER schema, identify the relations S and T that correspond to the entity types participating in R. Choose one of the relations – S, say – and include as foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role S. Include all the simple attributes of the 1:1 relationship type R as attributes of S.

For MLCA Integrated database, there are no 1:1 binary relations.

Step-4: For each regular (non-weak) binary 1:n relationship type R, identify the relations S that represents the participating entity type at the n-side of the relationship type. Include as a foreign key in S the primary key of the relation T that represents the other entity type participating in R; this is because each entity instance on the N-side is related to at most one entity instance on the 1-side of the relationship type. Include any simple attributes (or simple components of composite attributes) of the 1:n relationship type as attributes of S.

For MLCA Integrated database, there are no 1:n binary relations.

Step-5: For each binary m:n relationship type R, create a new relation S to represent R. Include as foreign key attributes in S the primary keys of the relations that represent the participating entity type; their combination will form the primary key of S. Also include any simple attributes of the m:n relationship type as attributes of S.

For MLCA Integrated database, the relation *Made_By* is created, as it is m:n relationship between the COMPONENT and MATERIAL. The primary keys of

COMPONENT and MATERIAL relations become foreign keys in *Made_by*. Similarly the relation *Used_by* is created which also represents m:n relationship between FACILITY and PROCESS. The primary keys of FACILITY and PROCESS relations become foreign keys in *Used_by*.

The relation *Processed_by* is created which also represents m:n relationship between MATERIAL and PROCESS. The primary keys of MATERIAL and PROCESS relations become foreign keys in *Processed_by*. The relation *Processed_by* also represents m:n relationship between COMPONENT and PROCESS. The primary keys of COMPONENT and PROCESS relations become foreign keys in *Processed_by*.

Step-6: For each multivalued attribute A, create a new relation R that includes an attribute corresponding to A plus the primary key attribute K (as a foreign key in R) of the relation that represents the entity type or relationship type that has A as an attribute. The primary key of R is the combination of A and K.

In MLCA integrated database, relation *Material_Alias* is created. The attributes of *Material_Alias* represents the multivalued attribute Alias of MATERIAL, while *Material_ID* as foreign key, represents the primary key of the MATERIAL relation. The primary key of *Material_Alias* is the combination of *Material ID, Alias*. A separate tuple will exist in *Material_Alias* for each alias associated with that material. Similarly for attribute *Composition_Material*, a relation *Material_Com_Mat* is created, and it has primary key as a combination of *Material_ID, Composition_material*.

In this way regular E-R is decomposed into relational schema. Now the next steps consider the enhanced E-R aspects.

Step-7: Convert each specialization with m subclasses $\{S_1, S_2, \ldots, S_m\}$ and (generalized) super-class C, where the attributes of C are $\{k, a_1, a_2, \ldots, a_n\}$ and k is the (primary) key, into relation schemas using one of the four options:

Option 7A: Create a relation L for C with attributes Attrs(L) = $\{k, a_1, a_2, \ldots, a_n\}$ and pk(L) = k. Create a relation L_i for each subclass S_i. 1<= I <= m , with the attributes Attrs(L) = {k} Union {attributes of S_i } and pk(L_i) = k.

Option 7B: Create a relation L_i for each subclass Si. 1<= I <= m , with the attributes Attrs(L) = {k} Union {attributes of S_i } and pk(L_i) = k.

Option 7C: Create a single relation L for C with attributes Attrs(L) = {k, a1, ..., an} union {attributes of S1} union .... Union {attributes of Sm} Union {t} and pk(L) = k. This option is specialization whose subclasses are disjoint, and t is a type attribute that indicates the subclass to which each tuple belongs, if any. This option has the potential for generating a large number of null values.

Option 7D: Create a single relation L for C with attributes Attrs(L) = $\{k, a_1, \ldots, a_n\}$ union {attributes of S1} union .... Union {attributes of Sm} Union $\{t_1, t_2, \ldots, t_m\}$ and pk(L) = k. This option is specialization whose subclasses are overlapping (not disjoint), and each ti, a<= I <= m, is boolean attribute indicating whether a tuple belongs to subclass Si.

Option 7A creates a relation L for the super-class C and its attributes, plus a relation Li for each subclass Si; each Li includes the specific attributes of Si, plus the primary key of the super-class C, which is propagated to Li and becomes its primary key.

An EQUIJOIN operation on the primary key between Li and L produces all the specific and inherited attributes of the entities in Si. Thus for the COMPONENT super-

class and its subclasses, an attribute *Comp_type* is created which describes whether COMPONENT is a PART or SUBASSEMBLY or FINAL-SUBASSEMBLY. New relations SUBASSEMBLY, FINAL-SUBASSEMBLY, PART and PRODUCT are created which will have their specific attributes. Relational tables for all entities are shown in tables 4.1 to 4.17.

Table 4.1 Relational Table for Entity COMPONENT

Component_ ID	Parent_ID	Name	Description	Color	Quantity
Weight	Unit_of_ Weight	Volume	Unit_of_Volume	Comp_Type	

Table 4.2 Relational Table for Entity PRODUCT

Component _ID	User_ Name	Company	Street	Zip	Telephone	Fax	Email	Date & Time

Table 4.3 Relational Table for Entity FINALSUBASSEMBLY

Component_ID	Material_ID	Weight	Unit_of_Weight

Table 4.4 Relational Table for Entity PART

Component_ID	Material_ID	Weight	Unit_of_Weight

Table 4.5 Relational Table for Entity CIRCUIT-BOARD

Component_ID	Area of Circuit Board

By same token, for super-class PROCESS and its subclass, an attribute *Process_type* is created which will inform the correct process.

Table 4.6 Relational Table for Entity PROCESS

Process_ ID	Material _ID	Component _ID	Name	Type	Process_ Time	Labor_ Time	Labor_ Rate_ Per_Hr	Yield_ rate

Table 4.7 Relational Table for Entity PRODUCTION Process

Process_ID	Facility_ID	Water_Input
Process_Id	Process_material_name	Weight_of process_material

Table 4.8 Relational Table for Entity USE Process

Process_ id	Active_ Mode_ %	Use_ Mode_ %	Idle_ Mode_ %	Power_ Save_ Mode %	Active_ Mode_ %	Use_ Mode_ %	Idl_ Mode %	Power_ Save_ Mode %

Table 4.9 Relational Table for Entity DEMANUFACTURING Process

Process _ID	Facilty _ID	Actual Rate	Pass Rate	Volume Handled	Disposal Fraction	Broken Part	Broken Link	Type_of_ Fastener	Tool_ Used

Table 4.10 Relational Table for Entity PACKAGING Process

Process_ID	Facilty_ID	% recycled	Total Weight per product

Process _ID	Material _Used	Weight of material per product

Table 4.11 Relational Table for Entity DISTRIBUTION Process

Process _ID	Facility_ID	Actual_Load	Load_Ratio	Distance_Traveled

Table 4.12 Relational Table for Entity MATERIAL EXTRACTION Process

Process_ID	Facility_ID	Number_of Steps

Process_ID	Step_Process_Name	Additive_name	Additive_weight	Water used

Table 4.13 Relational Table for Entity MATERIAL SYNTHESIS Process

Process_ID	Facility_ID	Number_of Steps

Process_ID	Step_Process_Name	Additive_name	Additive_weight	Water used

Similarly according to option 7A, for the superclass MLCA KEY FACTOR and subclasses POLLUTION INGREDIENTS, ENERGY and COST, the relational schema is as follows –

Table 4.14 Relational Table for Entity MLCA KEY Factors

MLCA_ID	Process_ID	Type of factor

Table 4.15 Relational Table for Entity POLLUTION INGREDIENTS

MLCA_ ID	WE_AL	WE_Ni	WE_Iron	WE_Hg	AE_ Ammonia	AE_ Aldehydes
WE_ Phenols	WE_ Lead	WE_ Cyanides	WE_ Ammnia	WE_ Sulphides	WE_ COD	WE_ Dissolved _Solids
WE_Oils	WE_BOD	WE_Acid	WE_ Chromium	WE_Metal _Ions	WE_ Suspended_ Solids	Solid_ Wastes
AE_other_ Organisms	AE_ Methane	AE_ Hydrocarbons	AE_Hg_ Vapour	AE_Gas_ Dust	AE_ Fluoride	AE_ Particulate
AE_Cl	AE_Nox	AE_Monoxide	AE_SO2	AE_CO	AE_CO2	AE_BOG

Table 4.16 Relational Table for Entity ENERGY

MLCA_ID	Energy_Value	Unit of Energy	Rate_Per_Hr	Total_ Hour

Table 4.17 Relational Table for Entity COST

MLCA_ID	Recovered_ Cost	Tooling_ Cost	Equipment_ Cost	Labor_ Cost	Maintenance_ Cost
Supplies_ Cost	Regrinding_ Cost	Process_ Cost	Scrap_ Cost	Virgin_ Cost	Resell_ Price
Power_ Cost	Other_ Cost				

4.4 Normalization

Normalization of data is defined as a process during which unsatisfactory relation schemas are decomposed by breaking up their attributes into smaller relational schemas that possess desirable properties. Thus normalization is a process to measure formally why one set of groupings of attributes into relation schema is better than another. Normalization simplifies the structure and eliminates data redundancies. Normal forms for relation schema have been defined, leading to progressively better groupings. Normal forms provide database designer with:

- A formal framework for analyzing relation schemas based on their keys and on the function dependencies among their attributes.
- A series of tests that can be carried out on individual relation schemas so that the relational database can be normalized to any degree. When a test fails, the relation violating that test must be decomposed into relations that individually meets the normalization tests.

The first Normal Form states that the domains of attributes must include only atomic (simple and indivisible) values and that the value of any attribute in a tuple must

be a single value from the domain of that attribute. Second Normal Form is based on the concept of full functional dependency while the third Normal form is based on transitive dependencies.

4.4.1 Boyce Codd Normalization

Boyce Codd Normal Form (BCNF) is another measure of normalization. It is stricter than the third normal form. A relation schema R is in BCNF if whenever a functional dependency x $\longrightarrow$ A holds in R, then X is a super-key of R.

For the integrated database every relation is checked for *Boyce Codd Normal form*. If we consider the relation COMPONENT, every attribute (*parent id, name, description, color, volume*) is functionally depend upon *Comp-ID* which is also a primary key thus satisfies Boyce codd Normal form. Similarly the entities PRODUCT, FINALSUBASSEMBLY, SUBASSEMBLY AND PART has all their special attributes depending upon attribute *Comp-ID* which is a primary key for these entities. Thus in this way all other relations can be checked for Boyce codd normal form.

This process of normalization strives to avoid anomalies, eliminate redundancies, simplify data retrieval, and maintain integrity.

CHAPTER 5

INTERFACE DESIGN

5.1 Integration of DEF and MLCA

Eco-efficiency has a significant impact in deciding the environmental policies and practices of major industries. The Business Council on Sustainable Development (BCSD) has introduced the concepts of eco-efficiency and suggested a link between resource efficiency (which leads to productivity and profitability) and environmental responsibility. Integration of DFE related disciplines, is a crucial step to successful development of eco-efficient products. By the same measures, an awareness of the product life cycle is critical to the practice of DFE [16].

As corporations becoming aware about the environmental aspects of a product in all stages of lifecycle, designers of the product have to consider end-of-life yield of assemblies, parts and materials while designing a product. Thus there is need to develop DFE tools using MLCA as a mechanism for quantifying not only the impact of design and production issues over the working life of a product but also the demanufacturing and reuse issues. To consider the environmental impact for all stages of lifecycle at the design stage requires bi-directional flow of information between design software and MLCA software. The Computer Aided Design (CAD) software contains the product level description. The MLCA software requires this product level information to perform the analysis of life cycle stages. The flow of information from CAD software to MLCA software is facilitated by the MLCA-CAD interface developed as part of this thesis research. The MLCA-CAD interface will provide the MLCA software the Bill of Material (BOM) information from CAD. After performing the MLCA analysis, the

designer of the product will receive environmental performance characteristics leading to products modifications which will make that product more eco-efficient. Thus the designer can make changes to the product in the CAD software and again the updated BOM information about the product will be supplied to the MLCA software using MLCA-CAD interface. This flow of information between CAD/DFE to MLCA is illustrated in figure 5.1. The other directional flow from MLCA software to CAD will be provided in terms of result of MLCA analysis and suggestions about modification of product via email to the user.

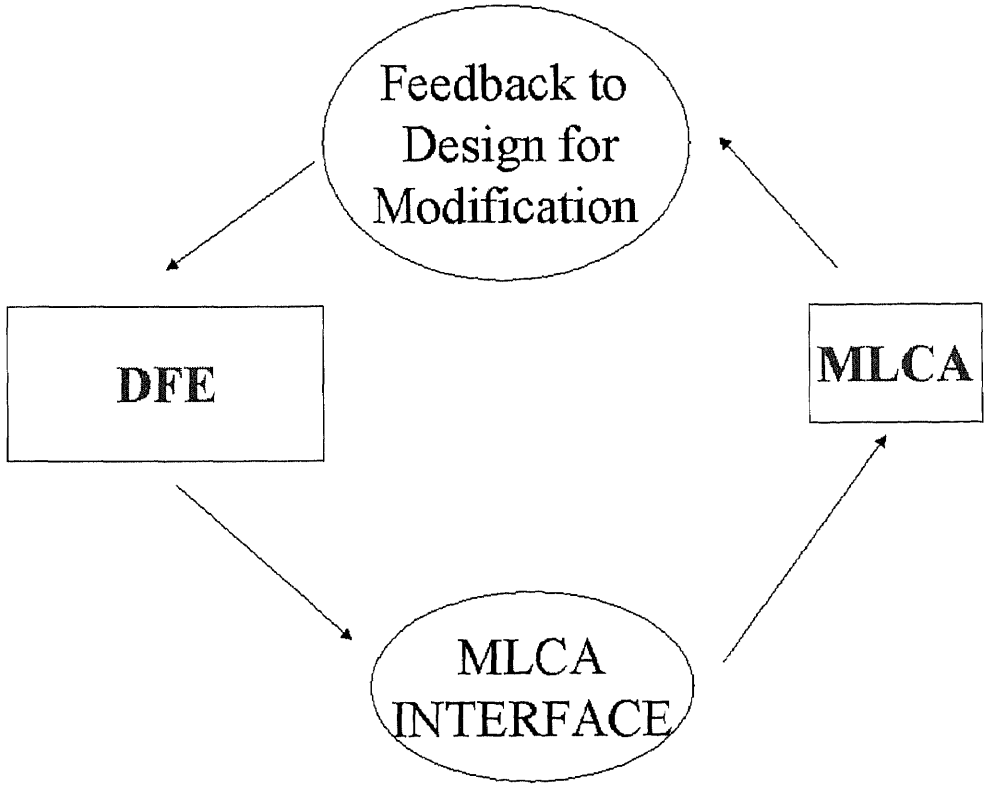


Figure 5.1 Integration Flow Diagram of MLCA and DFE.

5.2 Interface Requirements

As MLCA software is data centric, it requires a significant amount of data from the user. If the user has to manually populate the database, MLCA software will be cumbersome to user and potentially error-prone. As product description (BOM) data plays an important role in MLCA analysis, the main task of the MLCA-CAD interface is to populate the data from CAD package into the MLCA database programmatically. Thus this interface has a significant contribution to make MLCA more user friendly and less error prone while transferring data from CAD software to integrated database.

As described in chapter 4 the major users of the integrated database are the MLCA analyst, demanufacturer and product/process designer. Every user requires a clear understanding of product description for the analysis. The designer while designing populates the database of the CAD software. If the other users such as demanufacturers must enter the same product information, and without knowing the correctness of information, the analysis may be misleading. Since the interface imports data directly from CAD software, correct information specified by the designer is available to the MLCA analyst and demanufacturer, consequently everyone performs their analysis without requiring product information or constraints in detail. Thus MLCA-CAD interface serves the information sharing between different applications/users.

The features of MLCA-CAD interface are

- Reduce tedious manual input and thereby reduce potential for errors in data.
- Provide consistency in data and avoid redundancy.
- Makes data population fast, correct and broadly available.
- Easy to use.
- Version control of product design

5.3 Comparison of Different CAD Software

Over the past ten years, new design technologies have contributed to lower cost, higher quality products. Design tools have provided ways to automate and simplify their most basic tasks. These design tools, in turn, helped engineers to devote more time for creative elements of designing, and push the limits of the available technology. The more tasks the tools can successfully tackle, the more engineers expect from their tools. However it is not often that a single common need transcends all types of technical software users. Many issues that technical software users confront are specialized, pertaining only to a segment of the entire community [7].

For this thesis, the MLCA analysis focuses on electronic products. To demonstrate a case study, it requires to choose one of the available CAD software. The selection of software was made on the following basis – features, cost, functionality, ease of use, popularity and availability. According to popularity, the CATIA, I-DEAS, and Pro/ENGINEER software were considered for comparison.

CATIA is heavily used in the automotive and aerospace industries, with a customer base that includes Boeing, Lockheed Martin, Cessna, Mercedes Benz, BMW, Daimler Chrysler and General Dynamics [4]. Its core modeling extends customers ability to optimize digital product creation in machinery, consumer goods and consumer electronics industries. CATIA Generative Knowledge offers ways to generate intelligent designs automatically from scripts, enforcing best practices, for both improving design productivity and reducing errors [5]. Pro/ENGINEER is the de facto standard for mechanical design automation and is based on a parametric, feature-based, fully associative architecture. Pro/ENGINEER delivers a comprehensive suite of solutions for all areas of the development process, from a product's conceptual design and simulation

through manufacturing [6]. Feature-based, parametric, associative CAD systems have streamlined engineering by enabling the concurrent design of detailed models of all engineering deliverables that drive the design-through-manufacturing process [7]. I-DEAS Master Series 7 provides the functionality needed to create 2-D layouts and distribute those across teams. This allows for the 2-D layout to associatively change and update assemblies and their related part locations [8]. CATIA is more expensive than other software.

After the comparison of different CAD packages, Pro/ENGINEER was chosen as CAD package for developing MLCA-CAD interface, as it is the most commonly used CAD software in electronic products and also has a very good application programmer interface (API). Pro/TOOLKIT is the customization toolkit for Pro/ENGINEER from Parametric Technology Corporation (PTC.) It gives customers and third-party developers the ability to add functionality to Pro/ENGINEER by writing code in the 'C' programming language and integrating the resulting application into Pro/ENGINEER in a seamless way. Pro/TOOLKIT provides a large library of 'C' functions that enable the external application to access the database and applications of Pro/ENGINEER in a controlled and safe manner.

5.4 Interface Design

There are two steps in the development of the MLCA-CAD interface. The first is to develop a Pro/TOOLKIT application and next is to implement MLCA database population program. The steps of interface are illustrated in figure 5.2.

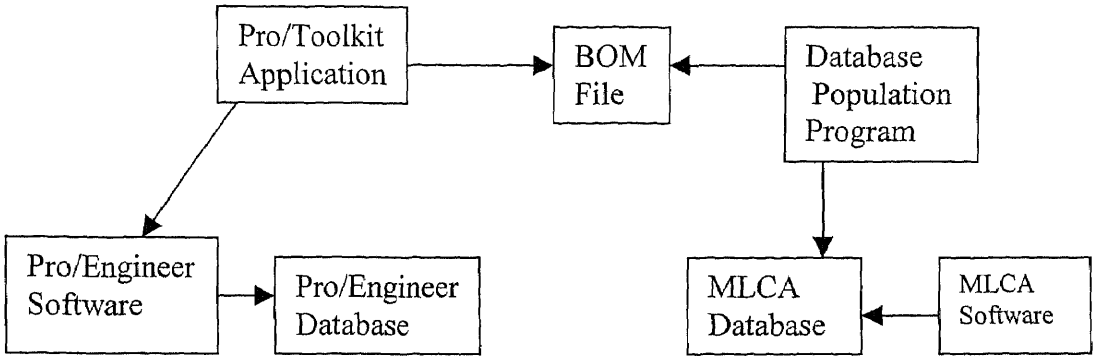


Figure 5.2 Design stages of MLCA-CAD interface

The Pro/TOOLKIT application program extracts the product information that is modeled in Pro/ENGINEER software, from the Pro/ENGINEER database. This application creates a flat file that contains the bill of material (BOM) information in a defined format. The second part contains an application 'Database Population Program', which connects to MLCA database and by reading the bill of material file, populates the MLCA database which is used by MLCA software.

5.4.1 Input Details

Although the main task of this interface is to reduce the amount of information input by the user, some user input will still be required. The user of the interface should download the Pro/TOOLKIT application for MLCA, which is available as a single bundle on the site http://www.merc.njit.edu/interface. The access to the application file is password protected. The user can apply for login to MERC at NJIT. User can download the Pro/TOOLKIT application. After downloading, the user can start Pro/ENGINEER release

2000I, as this interface is designed for Pro2000I version. Now the user can click on the menu #utilities and then under #utilities, on #Auxiliary applications, the screen shown in figure 5.3 will be displayed.

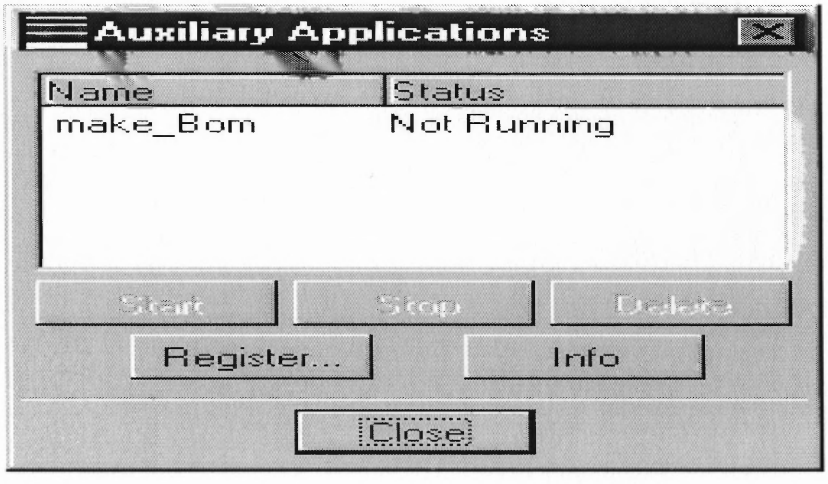


Figure 5.3 Startup Screen for Pro/Engineer for the interface

First the user is required to register the application by clicking on button 'Register'. Then to actually start the application, the user has to click on Start. This will create the file *MERC.XML*. The user has to again upload this *MERC.XML* file on MERC server by the screen shown in chapter 6 in figure 6.1.

5.4.2 Output Details

While grabbing the information from the product's assembly file opened in the Pro/ENGINEER, the interface will prompt the user for the missing information. Thus if the user has not entered any material for a part in that assembly, the application will prompt user to enter material information. Thus as a result of Pro/TOOLKIT application,

the newly created file *MERC.XML* will contain the information of the assembly and all of its sub-components, their name, their parent component/assembly name and quantity. If the component is a part, its material and weight is also written to the file.

After uploading the file on MERC server, the BOM file will be checked and the database population program will populate the database with the data in that BOM file. Thus, as the output of this interface, the user will have its product description data populated without any manual entry in MLCA software and will be ready to perform environmental analysis of the product on the MLCA software.

5.5 Implementation Details

5.5.1 BOM File Format

The core task in the implementation of the MLCA-CAD interface is defining the format of *MERC.XML* file, which will get created as a result of Pro/Toolkit application. This *MERC.XML* file should be programmatically readable. Hence the file is designed according to Standard Markup Language (XML) standard. Each record has a definite format and it is as follows.

<Component>

<Name>Actual Name of Component </Name>

<Type> 'ASM' for assembly and 'PRT' for part </Type>

<Material> Material of a part </Material>

<Weight> Weight of that part </Weight>

<Parent> Parent of a part/Subassembly. If it is a main product name, parent will be null</Parent>

</Component>

To add any new attribute to MLCA-CAD interface, one new entry will be added to the record format without disturbing the population program and thus this format is extensible.

5.5.2 Pro/toolkit Application Development

5.5.2.1 Pro/Toolkit Style: Pro/TOOLKIT uses a strongly object-oriented style in which the data structures that transfer information between Pro/ENGINEER and the application are not directly visible to the application, but are accessible only through the use of Pro/TOOLKIT functions. Pro/TOOLKIT also uses conventions for other aspects of style, for example strong typing, function prototyping, consistent and extensive function error status reporting, consistent function and data type naming conventions, use of enumerated types in preference to defines, and so on.

The most basic concepts in Pro/TOOLKIT are objects and actions. Each C function in the Pro/TOOLKIT library performs an action on an object of a specific type. The Pro/TOOLKIT convention for the name of each function is the prefix "Pro", the name of the object type, then the name of the action it performs. A Pro/TOOLKIT object is a well-defined and self-contained C structure that user can use to perform actions relevant to that object. Most objects are items in the Pro/ENGINEER database, such as features and surfaces. Others, however, are more abstract or transient, such as the information resulting from a select action. [21]

5.5.2.2 How Pro/Toolkit Works: The standard method by which Pro/TOOLKIT application code is integrated into Pro/ENGINEER is through the use of dynamically linked libraries(DLL). When Pro/TOOLKIT application 'C' code gets complied and

linked with the Pro/TOOLKIT library, an object library file gets created which is designed to link into the Pro/ENGINEER executable when Pro/ENGINEER starts up. This method is referred to as "DLL mode." [21]

5.5.2.3 Registering a Pro/Toolkit Application: Registering a Pro/TOOLKIT application means providing information to Pro/ENGINEER about the files that form the Pro/TOOLKIT application. To do this, create a small text file, called the Pro/TOOLKIT "registry file", that Pro/ENGINEER will find and read.

Pro/ENGINEER searches for the registry file in the following locations, in this order:

1. A file called protk.dat in the current directory
2. A file named in a PROTKDAT statement in the Pro/ENGINEER configuration file
3. A file called protk.dat in the directory <Pro/ENGINEER>/<MACHINE>/text /<LANGUAGE>
4. A file called protk.dat in the directory <Pro/ENGINEER>/text

In the last two options, the variables are as follows :

<Pro/ENGINEER>: The Pro/ENGINEER loadpoint (not the Pro/TOOLKIT loadpoint),

<MACHINE>: The machine-specific subdirectory (such as sgi_elf2 or i486_nt)

<LANGUAGE>: The name of the Pro/ENGINEER language the Pro/TOOLKIT application will be used with (such as "usascii" (English), "german", or" japanese")

If more than one registry file exists in this search path, Pro/ENGINEER finds them all and starts all the Pro/TOOLKIT applications specified in them. For this interface, a protk.dat file is provided to user [21].

5.5.2.4 Functions Used in Application: Pro/Toolkit provides a library of function and structures to access the data of an assembly or part and the application program developed in Pro/toolkit is saving that information in the file specified above. The following functions provided by pro/toolkit are used in this interface application program.

- ProMenuCreate() – The function ProMenuCreate() displays the menu on the screen. A menu must be loaded before it can be displayed; when a menu is redisplayed, all menu items are accessible.
- ProMenubuttonActionSet() - The function ProMenubuttonActionSet() specifies the action to perform when the user selects a particular menu button that has been added by the Pro/TOOLKIT application.
- ProMdlCurrentGet() - The function ProMdlCurrentGet() initializes the p_handle with the current Pro/ENGINEER object.
- ProAsmcompMdlGet() – The function ProAsmcompMdlGet() provides the 'ProMdl' handle to the part or assembly that is being referenced by the component.
- ProAsmcompTypeGet() – The function ProMdlTypeGet() provides the information to find out if the model is a part or an assembly.
- ProSolidFeatVisit() - Each component of an assembly is also a feature of that assembly. The function ProSolidFeatVisit() is used to visit the components, visit the features.

- ProMdlDataGet() - The function ProMdlDataGet() provides a C structure that contains information about the name and location of the operating system file in which the model is saved.
- ProSolidMassPropertyGet() - The function ProSolidMassPropertyGet() provides information about the distribution of mass in the part or assembly. It can provide the information relative to a coordinate system datum, which you name, or the default one if you provide NULL as the name. It returns the information in a structure called ProMassProperty that is declared in ProSolid.h.
- ProPartMaterialNameGet() - The ProPartMaterialNameGet() function retrieves the material assigned to the specified part.
- ProPartMaterialdataGet() - The ProPartMaterialdataGet() function retrieves the material properties for the specified part. The function returns the information in the ProMaterialProps data structure. [20]

The menu customization is implemented in file 'Bom.C'. When the 'MLCABOM" button is clicked, the function from file 'AsmCompPath.c' gets invoked. This function has implemented a recursive algorithm for traversing the assembly. In the recursive algorithm, the checking condition is to check against model type. If the model type is *part*, check for its material and mass properties, but if the model is an *assembly* traverse again all its sub-components. As for the functionality of the Pro/TOOLKIT application, the user should open the product assembly in Pro/ENGINEER package as the program takes the current assembly opened in Pro/Engineer. The details of code of files 'Bom.c' and 'AsmCompPath.c' can be found in Appendix B and Appendix C, respectively.

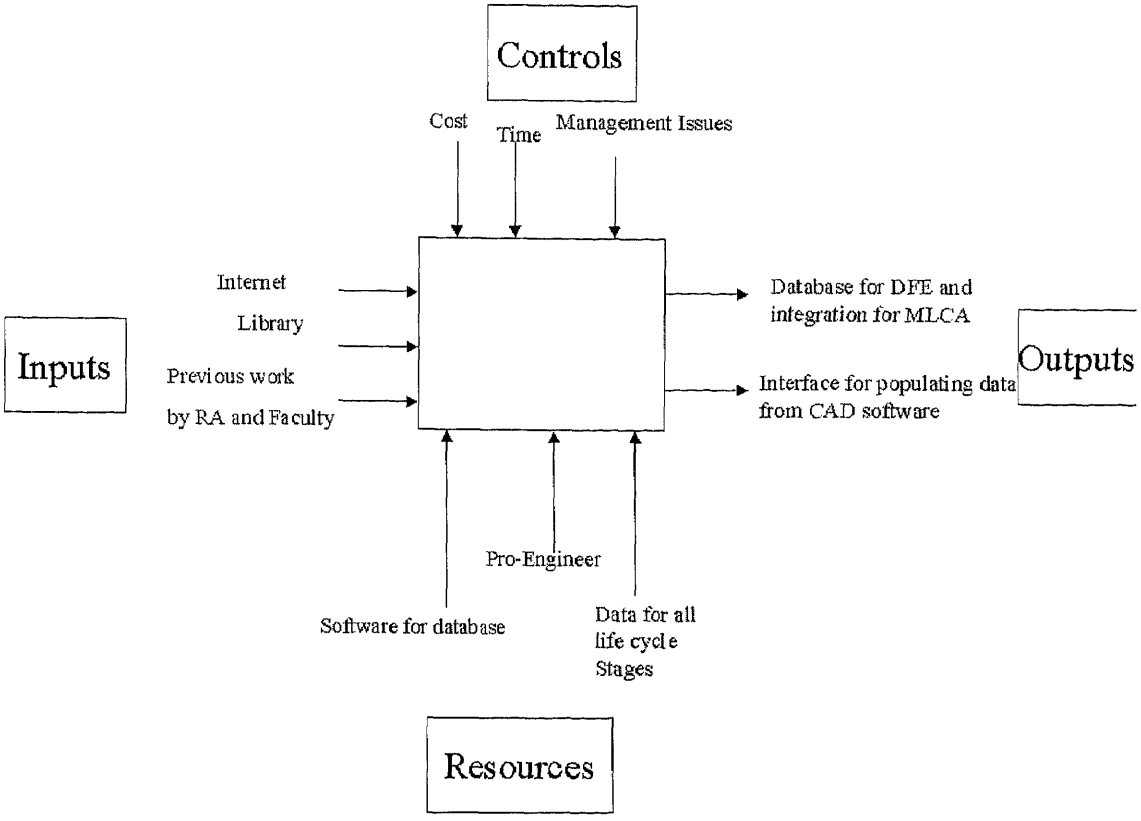


Figure 5.4 Process Object for Database Population Program

After creating the BOM file and uploading it on the MERC server, the data population program will run. The data population program is developed in JAVA and designed using object oriented concepts. The process object for this database population program is shown in figure 5.4. Internet, library, and the previous work done by research assistants are inputs to the design and development of database population program. The resources include the integrated database developed for MLCA, the BOM file generated by Pro/Toolkit application, and the database driver. The controlling forces are the cost and time.

The program uses Java Database Connectivity (JDBC) to connect to the database. The data population program updates the relations COMPONENT, PART, SUBASSEMBLY and PRODUCT and the data for that specific product will get populated. The version control is done at top level, viz., Product level. The code for the population program is included in Appendix D.

CHAPTER 6

CASE STUDY OF TELEPHONE –99

6.1 Introduction

As a case study, a typical *office telephone*, designed and manufactured in 1999 is considered as the MLCA-CAD interface application example. Using Pro/TOOLKIT application and the database population program, the MLCA database gets populated with the BOM information provided during designing/modeling in the Pro/ENGINEER CAD software. This chapter focuses on details of the steps followed to use the interface.

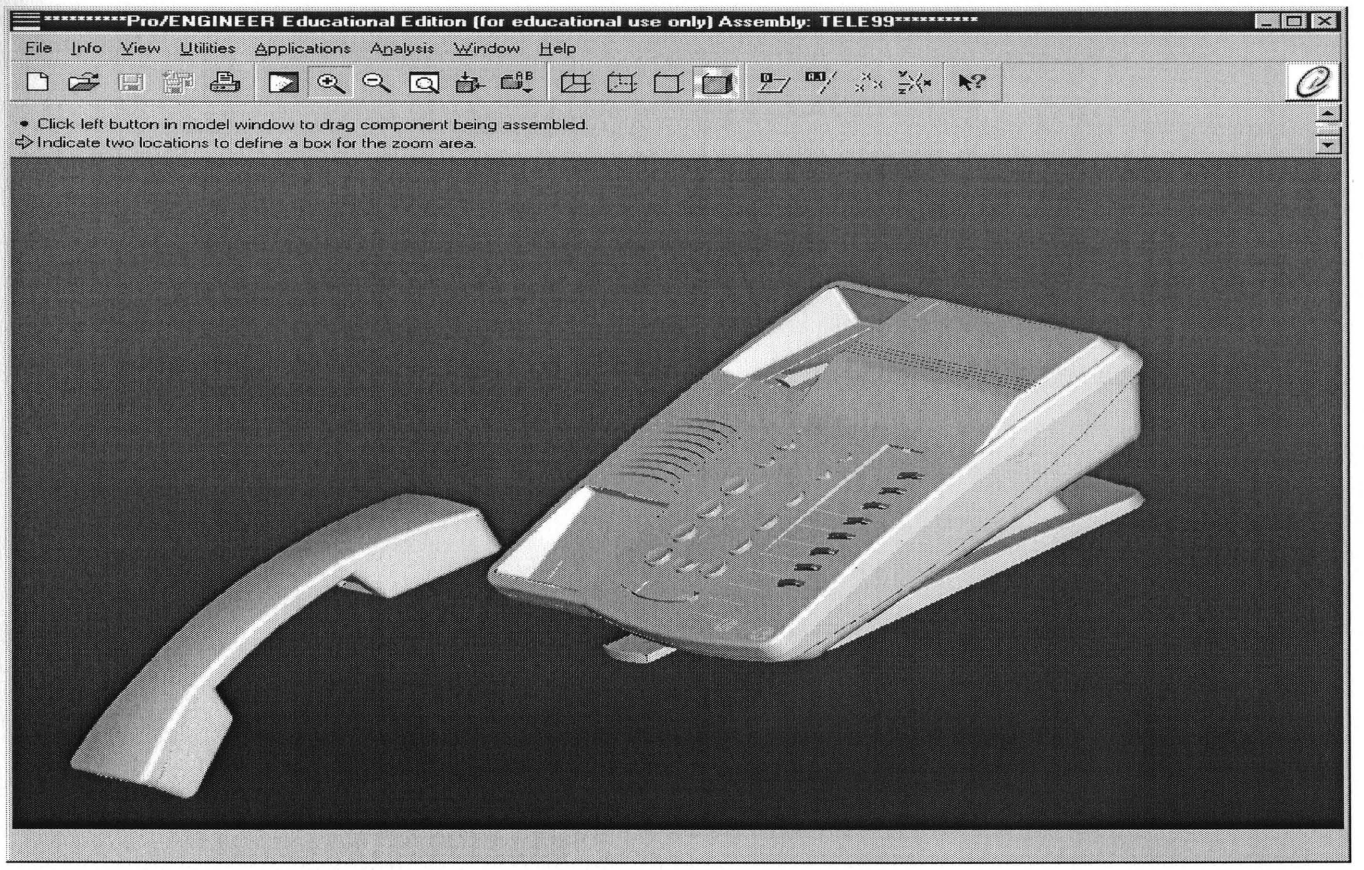


Figure 6.1 The *1999 Office Telephone* Assembly Modeled in Pro/Engineer Software

The prime objective of designing the integrated database is to provide decision-makers with information on environmental effects of a product and to identify opportunities for environmental improvements and thus eco-efficiency of that product. *1999 office telephone* is used here as a specific example to demonstrate how the information from Pro/ ENGINEER software gets populated to the integrated database. This *1999 office telephone* assembly model is created in Pro/ ENGINEER software and shown in figure 6.1. The photo of *1999 office telephone* is shown in figure 6.2.

Figure 6.2 The *1999 Office Telephone* Assembly Photo

6.2 Interface Usage

The interface is based on a client-server architecture. As described in Chapter 5, the MERC server provides the user with the ability to download the Pro/TOOLKIT application, from the web site *http://merc.njit.edu/prointerface.html*. The form shown in figure 6.3 will appear in user's browser.

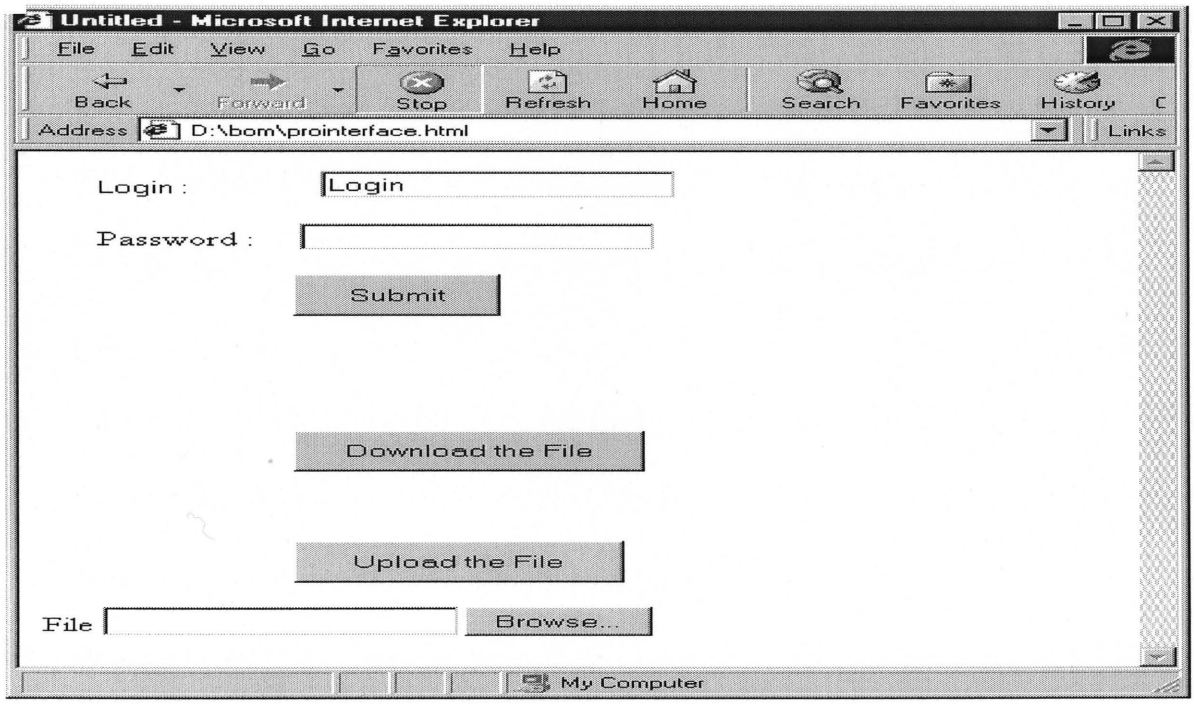


Figure 6.3 Pro/Toolkit Application Screen available at *http://merc.njit.edu /prointerface*

The Pro/TOOLKIT application is password protected. User requires a valid registration to use this interface program. To get a valid login and password, users can apply at http://merc.njit.edu. When the user provides valid login, the 'Download' and 'Upload' buttons become active. The user can then download the Pro/TOOLKIT application file by pressing the button 'Download'.

When the user presses the button for downloading the application, a zip file 'MLCA.zip' containing the files 'Bom.exe'(executable file), 'protk.dat' and a

'README' file, gets downloaded. The README file contains all the instruction for the user about how to start the Pro/TOOLKIT application. There is no need to have Pro/TOOLKIT module on the users machine as the application is unlocked. The README file focuses on the following instruction -

- Operating System – WINDOWS NT.
- Pro/Engineer Version 2000i.
- Uncompress the zip file 'MLCA.zip'.
- To register the 'Bom' application, user has to edit the file 'protk.dat' which is a part of the zip package. In that file under text_dir user has to write his own directory path.
- Run the Pro/ENGINEER from the same directory and the application will start by its own but if you start the Pro/ENGINEER from other directory, user has to register the application as shown in figure 5.1
- Once the application gets started, the user can find a new menu item added to Pro/ENGINEER default menu. The new menu item is between the menu *#Utilities* and *#Applications*. The menu is shown in figure 6.4. User can also notify that there is a message in the message window 'BOM for MLCA Software Creation Application'. If the user sees this message, that means the application started successfully.

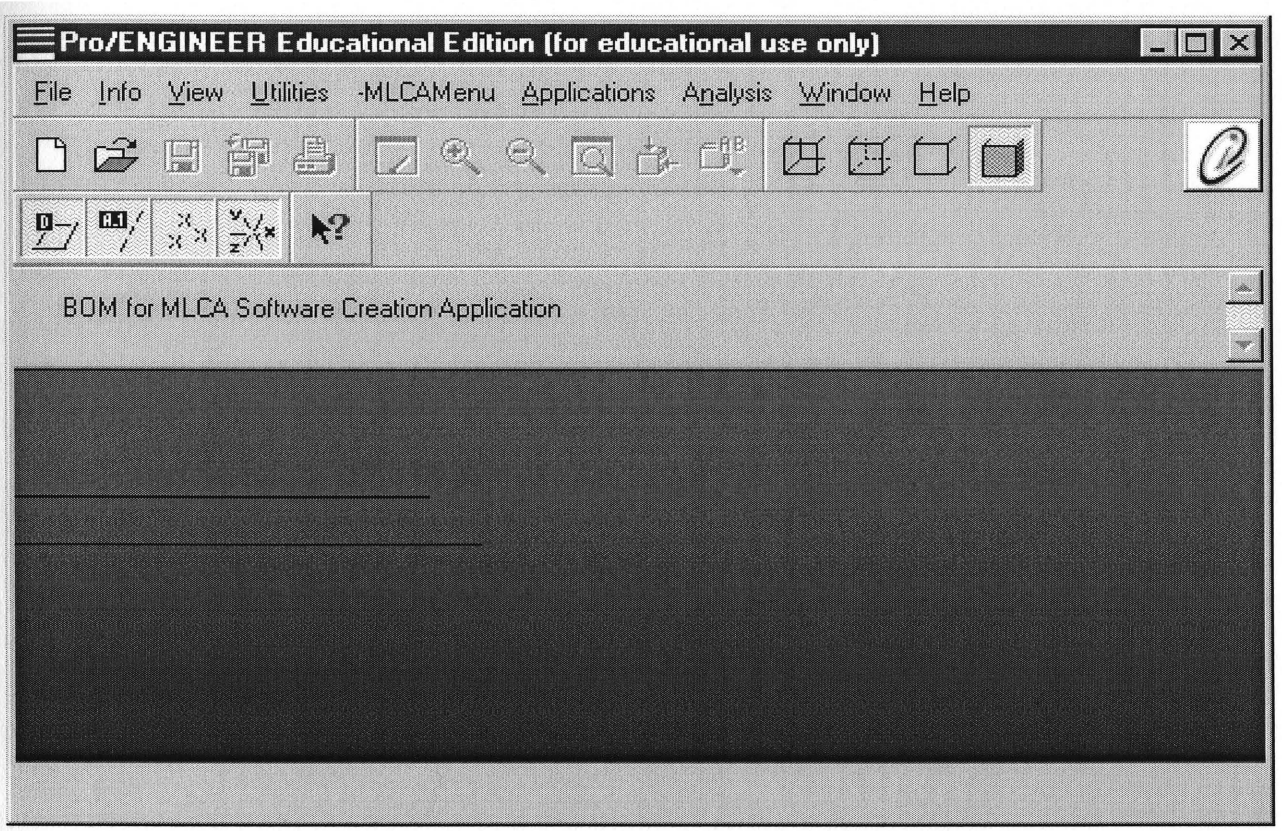


Figure 6.4 New Menu Added for MLCA Application

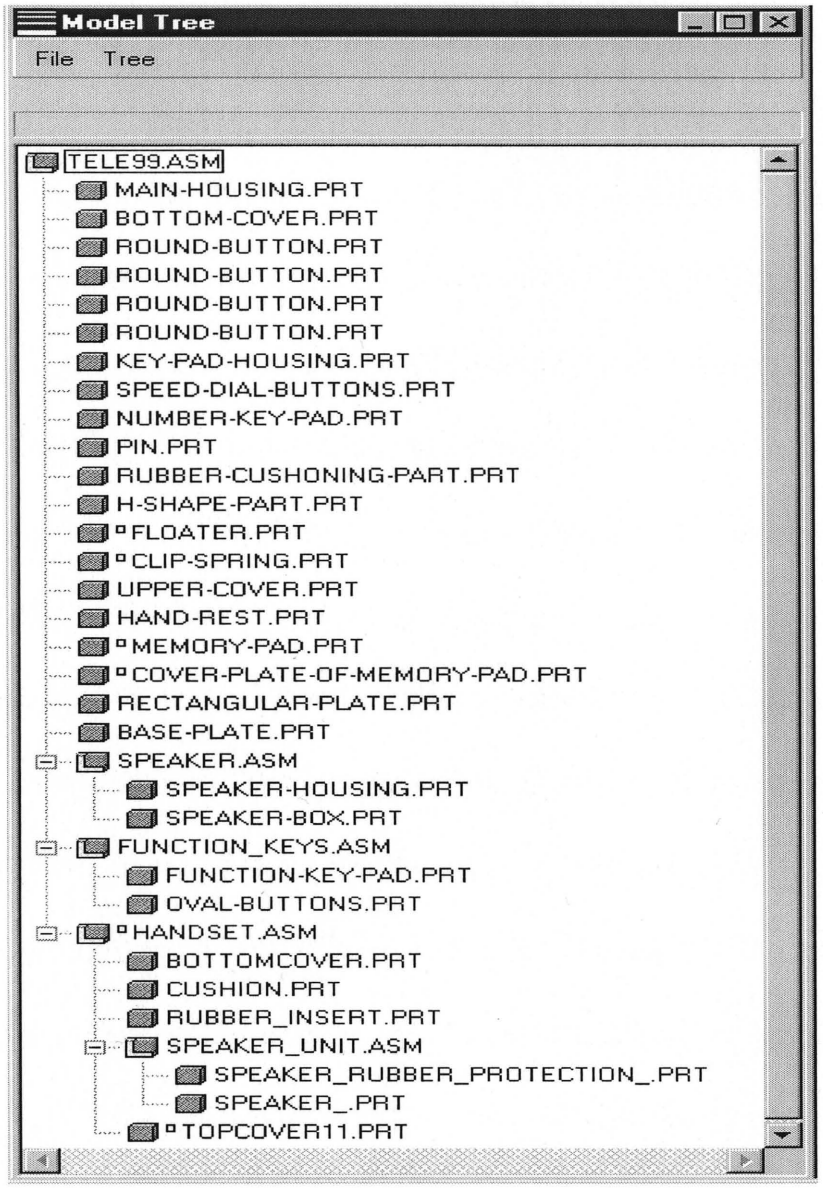
- User now has to open the product's assembly file for which he wants to create the Bill of Material file. After opening the assembly successfully, the user can start building the BOM for MLCA software by pressing the button #MLCABOM under the new menu #-MLCAMenu.
- If there is inadequate information in the assembly model, the program will prompt the user to enter the missing information. At the end, the file *MERC.XML* will get created and displayed to the user.

By following all the above instructions from the file README, the file *MERC.XML* gets created. The details of the file can be found in Appendix E. After

creating the file, the file can be uploaded back to the server by using the button 'Upload the File' shown in figure 6.3.

6.3 Database Population for Telephone 99

After uploading the file *MERC.XML*, the file is checked for the valid format defined in chapter 5. If the file is correct, the program for database population will run for that file. The *1999 office telephone* assembly tree in Pro/ENGINEER is shown in figure 6.5.



The *1999 office telephone* model viz. TELE99.ASM consists of subassemblies and parts. The subassemblies are 'SPEAKER.ASM', 'FUNCTION-KEYS.ASM' and 'HANDSET.ASM'. The subassembly 'SPEAKER.ASM' consists of parts 'SPEAKER-HOUSING.PRT' and 'SPEAKER-BOX.PRT' and the subassembly 'FUNCTION-KEYS.ASM' consists of parts FUNCTION-KEY-PAD.PRT' and 'OVAL-BUTTONS.PRT' as their children. The subassembly 'HANDSET.ASM' consists of one subassembly 'SPEAKER_UNIT.ASM' and parts 'BOTTOMCOVER.PRT', 'CUSHION.PRT', 'RUBBER_INSER.PRT', and 'TOPCOVER11.PRT'. The subassembly 'SPEAKER_UNIT.ASM' consists of two parts 'SPEAKER.PRT' and 'SPEAKER_RUBBER_PROTECTION.PRT' as its children. The remaining parts in the tree shown in figure 6.4 are children of assembly TELE99.ASM. This parent-child relationship is mapped in the database by using the uploaded *MERC.XML* file.

Pro/ENGINEER while assembling does not care about the quantity. In figure 6.3 the part 'ROUND-BUTTON.PRT' is shown four times in the tree instead of showing the quantity as four. The database population program check for the quantity by checking the name of component, and how many time the name appears in the tree, that number is considered as quantity of that component to perform the analysis.

The work of implementing the MLCA software with the integrated database is not completed yet. Currently the MLCA software is using the database designed by Ms. Jane Jin (Research Assistant at MERC) [17] Hence the database designed by Jane is used for populating the BOM information of data of *1999 office telephone* to carry out the MLCA analysis. The database population program was run using the file *MERC.XML*. The

database population program is implemented by standard XML parser provided by SUN Systems.

According to the current MLCA database structure the BOM information get stored in table PROJECT, PRODUCTION, RELATIONSHIP and PFS_MATERIAL. After running the database population program, the PROJECT table was updated and one new record was inserted in MLCA database for project as TELE99. The updated PROJECT table is shown in table 6.1. In the table PROJECT, the entry has been added by the database population program for TELE99 as *project name* and an attribute *project_ID* was assigned as 305 which is an automatic generated number by MS-Acess database. As *1999 office telephone* consists of three subassemblies and twenty seven parts, thirty records were inserted in table PRODUCTION for each subassembly and part. If the part has a parent as TELE99, the attribute *Identify_P2* is 'FP' and for the parts under any subassembly the *Identify_P2* is 'P'. The subassemblies are having the attribute *Identify_P2* value as 'S'. The updated PRODUCTION table is shown in table 6.2.

The table RELATIONSHIP stores the relationship between the parts under a subassembly with that subassembly. 12 records were inserted in table RELATIONSHIP for all parts and subassemblies under the subassembly. The updated table is shown in table 6.3. For the 27 parts, the material, quantity and weight were inserted in table PFS_MATERIAL shown in table 6.5. As a result of MLCA-CAD interface and database population program, the tree for TELE99 in MLCA software is shown in figure 6.4. The tree can be seen in Product Description stage of MLCA software. [17] Thus the MLCA database is ready to perform the multi-lifecycle assessment and to study environmental impacts.

Table 6.1 Table PROJECT after populating the data.

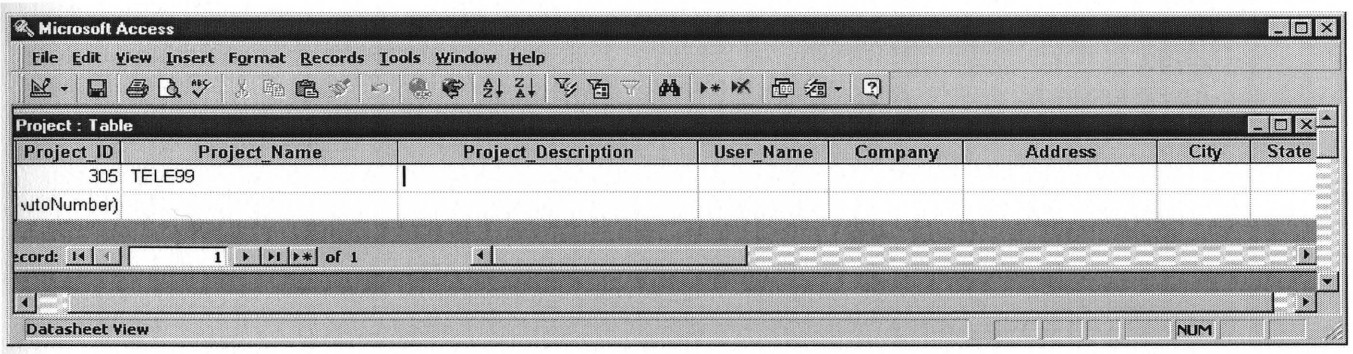
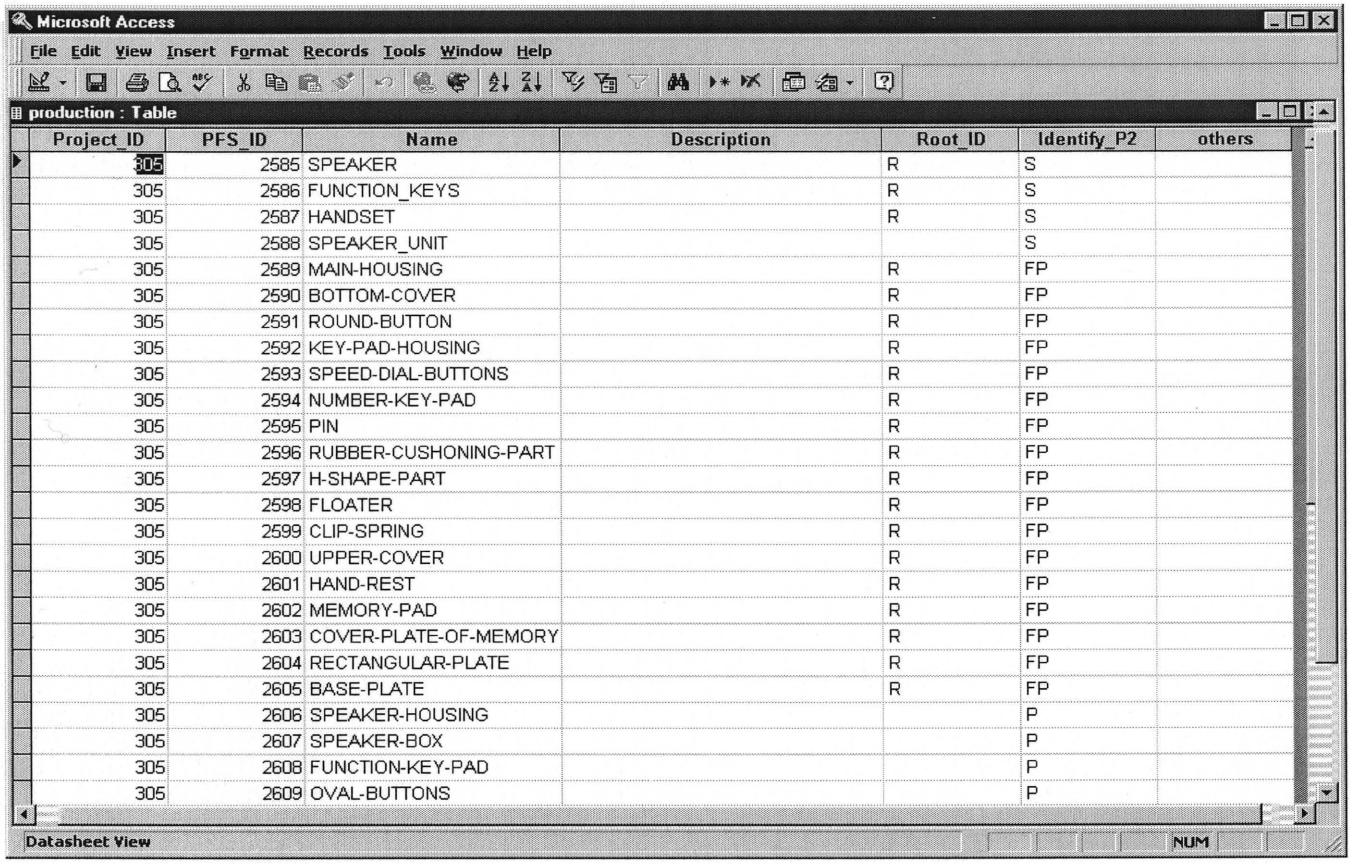


Table 6.2 Table PRODUCTION after populating the data.



Project_ID	PFS_ID	Name	Description	Root_ID	Identify_P2	others
305	2585	SPEAKER		R	S	
305	2586	FUNCTION_KEYS		R	S	
305	2587	HANDSET		R	S	
305	2588	SPEAKER_UNIT			S	
305	2589	MAIN-HOUSING		R	FP	
305	2590	BOTTOM-COVER		R	FP	
305	2591	ROUND-BUTTON		R	FP	
305	2592	KEY-PAD-HOUSING		R	FP	
305	2593	SPEED-DIAL-BUTTONS		R	FP	
305	2594	NUMBER-KEY-PAD		R	FP	
305	2595	PIN		R	FP	
305	2596	RUBBER-CUSHONING-PART		R	FP	
305	2597	H-SHAPE-PART		R	FP	
305	2598	FLOATER		R	FP	
305	2599	CLIP-SPRING		R	FP	
305	2600	UPPER-COVER		R	FP	
305	2601	HAND-REST		R	FP	
305	2602	MEMORY-PAD		R	FP	
305	2603	COVER-PLATE-OF-MEMORY		R	FP	
305	2604	RECTANGULAR-PLATE		R	FP	
305	2605	BASE-PLATE		R	FP	
305	2606	SPEAKER-HOUSING			P	
305	2607	SPEAKER-BOX			P	
305	2608	FUNCTION-KEY-PAD			P	
305	2609	OVAL-BUTTONS			P	

Table 6.3 Table RELATIONSHIP after populating the data.

Microsoft Access

File Edit View Insert Format Records Tools Window Help

Relationship : Table

ID	Root	Child	Project_ID	g
1063	0	305	305	0
1064	2587	2588	305	0
1065	2585	2606	305	0
1066	2585	2607	305	0
1067	2586	2608	305	0
1068	2586	2609	305	0
1069	2587	2610	305	0
1070	2587	2611	305	0
1071	2587	2612	305	0
1072	2587	2615	305	0
1073	2588	2613	305	0
1074	2588	2614	305	0
(AutoNumber)	0		0	0

Datasheet View NUM

Table 6.4 Table PFS_MATERIAL after populating the data.

Microsoft Access - [PFS_Material : Table]

File Edit View Insert Format Records Tools Window Help

PFS_ID	ID	Material_Name	Weight	Quantity	Project_ID	Unit	Material_ID	g
2589	1151	PLASTIC	235.585	1	305	gm	9	
2590	1152	PLASTIC	105.335	1	305	gm	9	
2591	1153	RUBBER	0.541	4	305	gm	11	
2592	1154	RUBBER	33.536	1	305	gm	11	
2593	1155	PLASTIC	10.931	1	305	gm	9	
2594	1156	PLASTIC	21.765	1	305	gm	9	
2595	1157	PLASTIC	0.248	1	305	gm	9	
2596	1158	RUBBER	1.103	1	305	gm	11	
2597	1159	PLASTIC	0.482	1	305	gm	9	
2598	1160	PLASTIC	5.09	1	305	gm	9	
2599	1161	STEEL	0.437	1	305	gm	213	
2600	1162	PLASTIC	202.624	1	305	gm	9	
2601	1163	PLASTIC	1.26	1	305	gm	9	
2602	1164	PAPER	0.382	1	305	gm	214	
2603	1165	PLASTIC	1.534	1	305	gm	9	
2604	1166	PLASTIC	14.846	1	305	gm	9	
2605	1167	PLASTIC	42.547	1	305	gm	9	
2606	1168	FOAM	36.797	1	305	gm	212	
2607	1169	FOAM	65.657	1	305	gm	212	
2608	1170	PLASTIC	6.952	1	305	gm	9	
2609	1171	PLASTIC	0.415	1	305	gm	9	
2610	1172	PLASTIC	69.395	1	305	gm	9	
2611	1173	FOAM	0.362	1	305	gm	212	
2612	1174	RUBBER	1.122	1	305	gm	11	
2613	1175	RUBBER	4.535	1	305	gm	11	
2614	1176	RUBBER	34.949	1	305	gm	11	
2615	1177	PLASTIC	47.683	1	305	gm	9	
	(AutoNumber)		0	0			0	

Record: 1 of 27

Datasheet View NUM

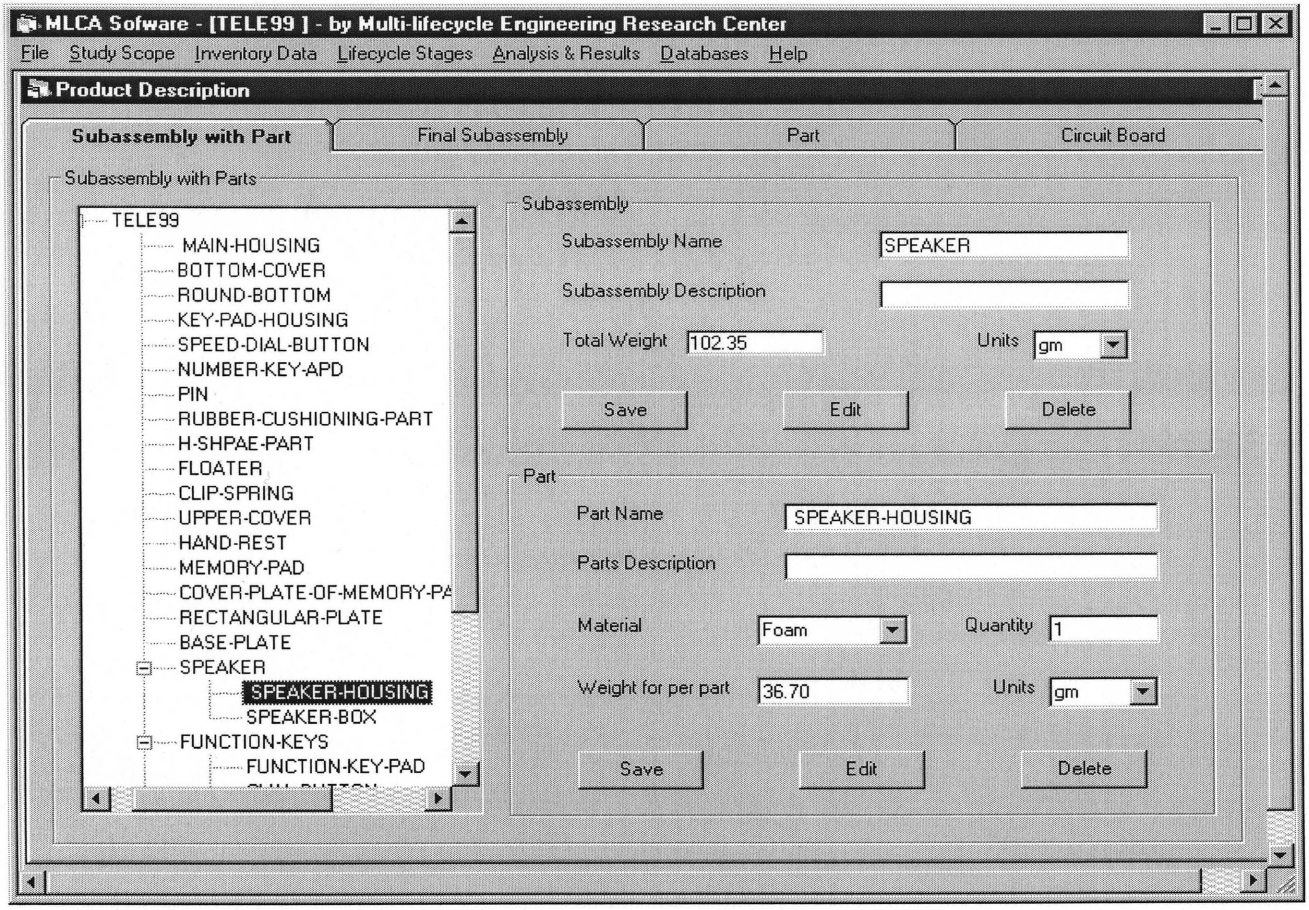


Figure 6.6 Product Description form of MLCA after populating the data

By using the MLCA-CAD interface, the BOM data for the *1999 office telephone* has been inserted in MLCA database in one mouse click. Thus it reduces the manual work and also eliminates errors that may occur by manually inserting the data, making MLCA software more user friendly, less tedious and less error prone.

CHAPTER 7

SUMMARY AND FUTURE WORK

7.1 Summary for Integrated Database

The integrated database for MLCA and DFE acts as a backbone for MLCA software. Designed using enhanced relation database concepts, the database embraces the features of relational and object oriented database management system. As MLCA itself is in early development, new relations, new entities, new attributes are evolving. The integrated database has the feature of extensibility as it implements object oriented database features.

7.2 General Conclusion

There are several issues related to the integrated database and interface and are summarized as follows:

- The traditional lifecycle approach is extended in MLCA to consider demanufacturing, remanufacturing, and reengineering stage [17].
- It is necessary to consider the environmental effects at the design stage. Hence it is very important to integrate the databases for MLCA and CAD for achieving DFE.
- The integration of Pro/ENGINEER to MLCA software is successfully carried out for the *1999 Office Telephone* that is described as a case study in chapter 5.

7.3 Future Scope and Research Recommendation

7.3.1 Integrated Database

Data collection is a main issue for the lifecycle assessment and their associative databases. To perform the multi-lifecycle methodology on a product and get accurate results through the analysis, data and information are needed on energy consumption, materials usage, and environmental burdens of each process in the entire product life cycle. Data and process information exists for several products, process and material [28]:

- The inventory tables of CRT, TV, PC computer, and four generations of office telephones.
- The energy consumption and environmental burdens generated from the production of such feedstock materials as plastics, metals, and glass.
- Disassembly data on shredding operation, and
- Reengineering polymers.

The resources of the information are from different sites, including the public free data on the web, technical books, research papers and publications. The problems of data item are uncertainty and data distribution. One solution is to associate a data item with such values as its mean, deviation, and certainty. The data still under search includes [17]:

- Process data such as cleaning process in reengineering and remanufacturing stages.
- Energy requirement in processes.
- Demanufacturing data.

Efforts should be taken to analyze the MLCA stages in depth so that new entities and attributes will get added to the integrated database. Efforts should be taken to analyze the size of the data, as data needs may increase significantly. It is also necessary to make connection with public databases and develop interfaces for database populations for those public databases.

7.3.2 MLCA-CAD Interface

As discussed above, MLCA is a continuously growing concept, and hence the new entities are evolving. So it is likely to extend the MLCA-CAD interface program for getting more information from CAD software. Currently there is direct flow of information from Pro/ENGINEER to MLCA but not the reverse way. So the Interface program can be extended to take the flow from MLCA to Pro/Engineer into consideration.

This program is developed for the operating system WINDOWS-NT only. Changes have to be made to run on the different operating System. In the latest version of Pro/ENGINEER, a feature *Jlink* has been added. Using the feature *Jlink,* MLCA-CAD interface program can be written in JAVA to make it independent of operating system and thus making the interface program portable.

This interface is currently working for Pro/ENGINEER software only. But different electronic industry uses different CAD software. So this interface design can be implemented for other CAD software by studying their application program interface. (API)

7.3.3 MLCA Software

Currently the MLCA software is developed in Microsoft Visual Basic 6.0 and hence it is operating system dependent. The CAD softwares are available on many platforms. If MLCA software is implemented in JAVA, then it can run on various platforms without recompilation. It can use Java Database Connectivity (JDBC) to access the common repository for components where all modeling information is stored. Figure 7.1 shows the model of the next version of MLCA software, which is fully integrated with the Pro/ENGINEER software.

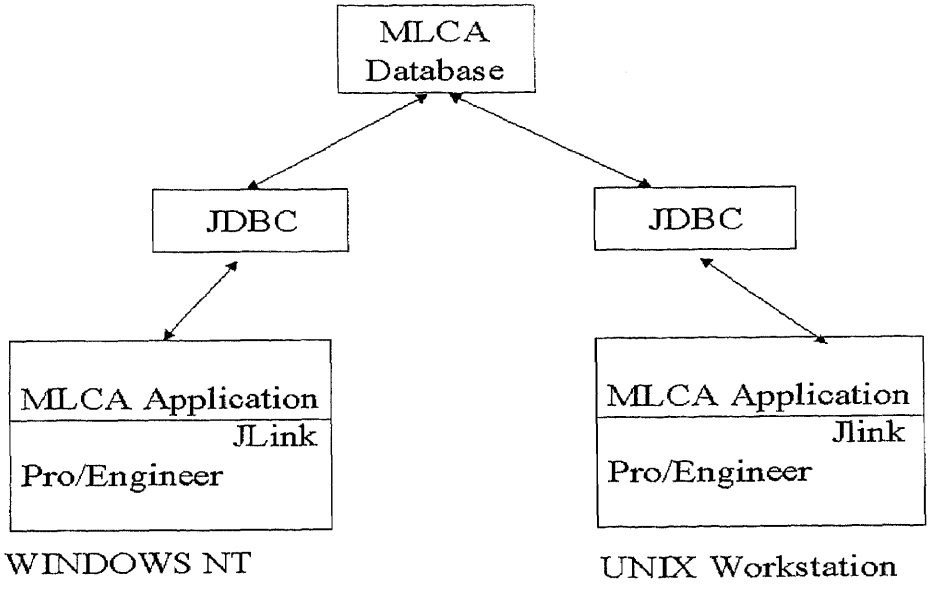


Figure. 7.1 Future Model of MLCA software

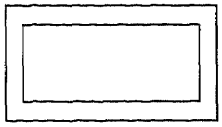
By using the *Jlink* feature of Pro/ENGINEER, the interface program on any remote machine will populate the database for MLCA on MERC server using JDBC. After populating the MLCA database, the interface implemented using *Jlink*, can invoke the MLCA JAVA application. Then all the forms of MLCA software will appear on Pro/ENGINEER screen and user can analyze the MLCA analysis result in the Pro/Engineer Software. The MLCA application will access the information stored in the MLCA database for performing the MLCA analysis, using JDBC. The result of MLCA analysis will get stored back into MLCA database. The same application will run for both WINDOWS NT and UNIX workstation.

APPENDIX A

DATABASE ENTITY-RELATION DIAGRAM SYMBOLS

1. Entity type/class Symbols

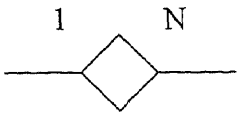
2. Weak Entity Symbol



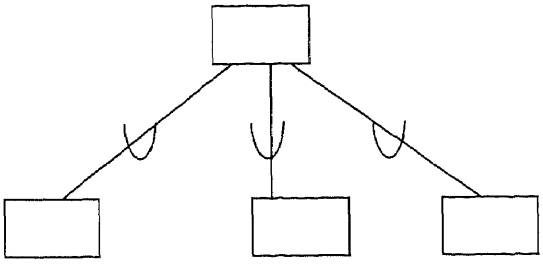
3. Attribute Symbol

4. Relationship Symbol

5. Cardinality Ratio Symbol



6. Specialization Symbol



APPENDIX B

SOURCE CODE FOR FILE BOM.C

```
/** File implementing menu custamization functions
  * @Author Bhagyashree Suratran
  * Version 1.0 at Nov 1, 99
*/

/* Including file from Pro/Toolkit Library
*/
#include <ProToolKit.h>
#include <ProMessage.h>
#include <TestError.h>
#include <ProMdl.h>
#include <ProMenu.h>
#include <ProUICmd.h>
#include "ProMenuBar.h"
#include <ProFeature.h>
#include <ProObjects.h>
#include <TreeTypesTable.h>
#include <ProObjects.h>
#include <ProMaterial.h>
#include <ProSolid.h>

/* Including files created by applicants
*/
# include "AsmCompVisit.c"

/** method defined in file AsmCompVisit.c
 * returns 1 for successfull or 0 for failed
*/
extern int UserAsmCompVisit();

/* Method main to initialize the Pro/Toolkit function
*/
int main (int argc,char *argv[])
{
   fprintf (stderr,"BOM Started \n");
   ProToolkitMain(argc,argv);
   return 0;
}

/* Method to define the access mode
*/
static uiCmdAccessState TestAccessDefault(uiCmdAccessMode access_mode)
{
   return (ACCESS_AVAILABLE);
}

/* Method user_initialize starts execution
 * implements Menu Custamization call
 * defines tha action under the button MLCABOM
*/
int user_initialize ()
```

```
{
   uiCmdCmdId cmd_id;
   ProError status;
   ProFileName UserMsg;
   ProStringToWstring (UserMsg, "msg_bom.txt");
   ProMessageDisplay(UserMsg, "USER %0s", " BOM for MLCA Software
Creation Application");

   fprintf (stderr,"initialized Started \n");
   status = ProMenubarMenuAdd ("MLCAMenu", "USER -MLCAMenu",
"Utilities", PRO_B_TRUE, UserMsg);

   status = ProCmdActionAdd ("UserDispMsg",
(uiCmdCmdActFn)UserAsmCompVisit
, uiCmdPrioDefault, TestAccessDefault, PRO_B_TRUE, PRO_B_TRUE,
&cmd_id);

   status = ProMenubarmenuPushbuttonAdd("MLCAMenu", "MLCABom", "USER -
MLCABom", "USER New Button Help.", NULL, PRO_B_TRUE, cmd_id, UserMsg);

   fprintf (stderr,"initialized stoped \n");

   return 0;
}

/* Method user_terminates to stop the execution
*/
void user_terminate ()
{
}
```

APPENDIX C

SOURCE CODE FOR FILE ASMCOMPVISIT.C

```
/* File implementing recurssive alogorithem to traverse the assemblies
 * and their children
 * @Author Bhagyashree Suratran
 * version 1.0 at Nov 1, 99
*/


/* File included from Pro/Toolkit library
*/
#include <ProToolkit.h>
#include <ProMdl.h>
#include <ProFeature.h>
#include <ProObjects.h>
#include <ProFeatType.h>
#include <TestError.h>

/* Global definations
*/
#define FILENAME "MLCA.xml"
char assemblyName[PRO_NAME_SIZE];

typedef struct user_appdata
{
    FILE *fp;
    int level;
    char parent[PRO_NAME_SIZE];
} UserAppdata;

/* Method Prototype
*/
int  UserAsmCompVisit(void *dummy, int dummy2);
ProError UserAsmCompFilter(ProFeature *feature,ProAppData app_data);
ProError user_action(ProFeature *feature,ProError status,ProAppData
appdata);
int getPartProp(ProMdl p_part, FILE *fp);

/* Method user_action implements file writing
 * returns ProError if there is nay error
*/
int flag = 0;
char lastAssemblyName[PRO_NAME_SIZE];
ProError user_action(
    ProFeature *feature,
    ProError status,
    ProAppData appdata)
{
    FILE *fp;
    int l,level,res;
    char parent[PRO_NAME_SIZE];
    ProError err;
    ProMdl mdl;
```

```
    char name[PRO_NAME_SIZE];
    char type[PRO_TYPE_SIZE];
    wchar_t wname[PRO_NAME_SIZE];
    UserAppdata *appd,appd1;
    ProMdldata mdldata;


    appd = (UserAppdata *)appdata;
    fp = appd->fp;
    level = appd->level;
    strcpy(parent, appd->parent);
    err = ProAsmcompMdlGet(feature , &mdl);
    ERROR_CHECK( "user_action", "ProAsmcompMdlGet", err );
    err = ProMdlDataGet(mdl, &mdldata);
    ERROR_CHECK( "user_action", "ProModelitemNameGet", err );
    err = ProWstringToString(name,mdldata.name);
    err = ProWstringToString(type,mdldata.type);


    for(l=0;l<level;l++)
            fprintf(fp,"    ");
    fprintf(fp,"<Component> <Type type = \" %s \" /> <Name name =
\"%s\" /> <Parent parent= \" %s\" />",type, name, parent);
    if (strncmp(type,"PRT",3) == 0)
    {
        res = getPartProp(mdl, fp);
    }
    if (strncmp(type,"ASM",3) == 0)
    {
        fprintf(fp, "</Component>\n");
        appd1.fp = appd->fp;
      appd1.level = appd->level+1;
        strcpy(appd1.parent, name);
        err = ProSolidFeatVisit(mdl, user_action, UserAsmCompFilter,
&appd1);
      ERROR_CHECK( "user_action", "ProSolidFeatVisit", err );
    }
    if (feature != NULL) return(PRO_TK_NO_ERROR);
    return(PRO_TK_CONTINUE);
}

/* Method to write out the members of the current assembly, and display
 * the result in an information window.
*/
int UserAsmCompVisit(void *dummy, int dummy2)
{
    char name[PRO_NAME_SIZE];
    char type[PRO_TYPE_SIZE];
    wchar_t wname[PRO_NAME_SIZE];
    ProMdldata mdldata;
    ProError err;
    UserAppdata appdata;
    FILE *fp;
    ProMdl p_asm;

    err = ProMdlCurrentGet(&p_asm);
    ERROR_CHECK( "UserAsmCompVisit", "ProMdlCurrentGet", err );
```

```
/*
    Open the text file.
*/
    strcpy(name,FILENAME);
    fp = fopen(name,"w");
    err = ProMdlDataGet(p_asm, &mdldata);
    ERROR_CHECK( "UserAsmCompVisit", "ProMdlDataGet", err );
    ProWstringToString(name, mdldata.name);
    ProWstringToString(type, mdldata.type);
    strcpy(assemblyName,"NULL");
    fprintf(fp, "<?xml version=\"1.0\"?>\n");
    fprintf(fp, "<start>\n");
    fprintf(fp,"<Component> <Type type = \" %s\" /> <Name name= \" %s\"
/> <Parent parent= \"%s\" /></Component>\n",type, name,assemblyName);

    appdata.fp = fp;
    appdata.level = 1;
    strcpy(appdata.parent, name);
/*
    List the assembly members
*/
    err = ProSolidFeatVisit(p_asm, user_action, UserAsmCompFilter,
&appdata);
    ERROR_CHECK( "UserAsmCompVisit", "ProSolidFeatVisit", err );
/*
    Close the file, and display it.
*/
    fprintf(fp, "</start>");
    fclose(fp);
    ProStringToWstring(wname, FILENAME);
    err = ProInfoWindowDisplay(wname, NULL, NULL);
    ERROR_CHECK( "UserAsmCompVisit", "ProInfoWindowDisplay", err );
    return(PRO_TK_NO_ERROR);
}

/* Method UserAsmCompFilter()
 * implementing  A filter used by ProSolidFeatVisit() to visit
 * features which are assembly components
*/
ProError UserAsmCompFilter(
    ProFeature *feature,
    ProAppData app_data)
{
    ProError status;
    ProFeattype ftype;

/*
    Get the feature type
*/
    status = ProFeatureTypeGet(feature, &ftype);

/*
    If the feature is an assembly component,
        return NO ERROR,
    else
        return CONTINUE
*/
```

```
    if(ftype == PRO_FEAT_COMPONENT)
        return(PRO_TK_NO_ERROR);
    return(PRO_TK_CONTINUE);
}


/* Method getPartProp()
 * write the propertied such as material and weight to the file
*/
int getPartProp(ProMdl p_part, FILE *fp)
{
   char name[PRO_NAME_SIZE];
   ProError err;
   ProMaterial mtl;
   ProMaterialdata mtl_data;
   ProMassProperty mass_prop;

   err = ProMaterialCurrentGet(p_part, &mtl);
   ERROR_CHECK( "MyPartPropn", "ProMaterialGet", err);
   err = ProMaterialDataGet(&mtl, &mtl_data);
   err = ProSolidMassPropertyGet(p_part, NULL, &mass_prop);
   ProWstringToString(name, mtl.matl_name);
   fprintf(fp, " <Material material = \" %s \" /> <Weight weight=\" %f
\" /> </Component>\n ", name, mass_prop.mass);
   return 0;

}
```

APPENDIX D

SOURCE CODE FOR DATABASE POPULATION PROGRAM

The Database Population Program include the source code for classes –

- Assembly
- AssemblyList
- Part
- PartList
- XMLRead
- MLCADriver

```
/** Class assembly Implements component as assembly
  * @Author - Bhagyashree Suratran
  * @Version - 1.0 of Nov 22, 1999
*/

class assembly
{
   String name;
   String type;
   String parent;
   int quantity;
   int compID;
   int parentID;

   /*Constructor for class assembly*/
      assembly( String n, String p)
   {
      name = new String(n);
        parent = p;
        //If the parent is null it is the main product
        if(p.trim().equalsIgnoreCase("NULL"))
           type = new String("product");
        else
           type = new String("S");
        quantity = 1;
   }

   /*Method printData for printing the assembly data
     * @returns void       */
   public void printData()
   {
      //System.out.println(this.name+" "+this.compID+" "+this.type+"
"+this.parent+" "+this.parentID+" "+this.quantity);
      System.out.println(this.compID+"     "+this.parentID);
   }
}
```

```
/** Class assemblyList Implements a list for assembly
  * @Author - Bhagyashree Suratran
  * @Version - 1.0 of Nov 22, 1999
*/


import java.io.*;
import java.util.*;
import org.w3c.dom.*;
import com.sun.xml.tree.*;

class AssemblyList
{
   Vector AsmList;

   /*Constructor for class AssemblyList*/
   AssemblyList()
   {
      AsmList = new Vector();
   }

   /* Method addAssembly require assembly as 'a' to insert into vector
    * returns void
   */
   public void addAssembly(assembly a)
   {
      try
        {
         AsmList.addElement(a);
      }
        catch(Exception e)
        {
           System.out.println("exception in adding in assembly vector:
"+e.getMessage());
        }
   }

   /* Method isPresent require an assembly name as a string 'n'
    * returns assembly if present else null
   */
   public assembly isPresent(String n)
   {
      for(int i=0; i<AsmList.size(); i++)
        {
            assembly a =(assembly)AsmList.elementAt(i);
            if(a.name.equalsIgnoreCase(n))
                 return a;
        }
        return null;
   }

   /* Method updateQuantity require assembly  as 'a' to increment the
qunatity by 1
    * returns null
   */
   public void updateQuantity(assembly a)
   {
```

```
        a.quantity +=1;
    }

    /* Method printAssembly print all assembly in the list
     * returns null
    */
    public void printAssembly()
    {
       System.out.println("Printing assemblies");

       for(int i=0; i<AsmList.size(); i++)
         {
             assembly a =(assembly)AsmList.elementAt(i);
             a.printData();
         }
    }

    /* Method getSize of assembly list
     * returns size of assembly list
    */
    public int getSize()
    {
        return AsmList.size();
    }

    /* Method getAssemblyAt requries the index as 'i'
     * returns assembly at position i in the list
    */
    public assembly getAssemblyAt(int i)
    {
       return (assembly)AsmList.elementAt(i);
    }

    /* Method getParent of requires a string as 'parentName'
     * returns compID as parent's compID else zero
    */
    public int getParent(String parentName)
    {
          for(int i=0; i<AsmList.size(); i++)
          {
                   assembly a =(assembly)AsmList.elementAt(i);
                   if
(a.name.trim().equalsIgnoreCase(parentName.trim()))
                    {
                    return a.compID;
                    }
          }
          return 0;
    }

    /* Method assignParentID assigns each assembly a parentID by
checking the parent Name
     * returns void
    */
    public void assignParentID()
    {
          for(int i=0; i<AsmList.size(); i++)
```

```
        {
                assembly a =(assembly)AsmList.elementAt(i);
                if(i == 0)
                {
                      a.parentID = 0;
                }
                else
                   a.parentID = getParent(a.parent);
        }
    }

    /* Method addID requires a number as 'startID'
     * assigns each assembly in the list compID as the consequative
number
     * starting from startID returns the next highest number
     * used onlr for new MLCa database develoed by Bhagyashree Suratran
     */
    public int addID(int  startID)
    {
         for(int i=0; i<AsmList.size(); i++, startID++)
         {
             assembly a =(assembly)AsmList.elementAt(i);
             if(a.parent.equals("NULL"))
             {
                     a.compID = startID;
                     a.parentID = 0;
               }
               else
               {
                     a.compID = startID;
                     a.parentID = getParent(a.parent);
             }
       }
         return startID;
    }

}
```

```
/** Class part Implements component part
  * @Author - Bhagyashree Suratran
  * @Version - 1.0 of Nov 22, 1999
*/

class part
{  String name;
   String type;
   String parent;
   String material;
   double weight;
   int quantity;
   int compID;
   int parentID;

   /*Constructor for class part*/
   part(String n, String p, String m, double w)
   {
      name = n;
      parent = p;
      type =new String ("p");
      material = m;
      weight = w;
      quantity = 1;
   }

   /*Mathod printData prints the data of part
     @returns void */
   public void printData()
   {
      //System.out.println(this.name+" "+this.compID+" "+this.type+"
"+this.parent+" "+this.parentID+" "+this.material+" "+this.weight+"
"+this.quantity);
      System.out.println(this.compID+"          "+this.parentID);
   }
}
```

```
/** Class PartList Implements a list for part
  * @Author - Bhagyashree Suratran
  * @Version - 1.0 of Nov 22, 1999
*/
import java.io.*;
import java.util.*;
import org.w3c.dom.*;
import com.sun.xml.tree.*;

class PartList
{
   Vector PartList;
   /*Constructor for class PartList*/
   PartList()
   {
         PartList = new Vector();
   }

  /* Method addPart require part as 'a' to insert into vector
   * returns void
   */
   public void addPart(part a)
   {
       PartList.addElement(a);
   }

   /* Method isPresent require a part name as a string 'n'
    * returns part if present else null
    */
   public part isPresent(String n)
   {
       for(int i=0; i<PartList.size(); i++)
        {
            part a =(part)PartList.elementAt(i);
            if(a.name.equalsIgnoreCase(n))
                 return a;
        }
        return null;
   }

   /* Method updateQuantity require part as 'a' to increment the
qunatity by 1
    * returns null
    */
   public void updateQuantity(part a)
   {
      a.quantity +=1;
   }

   /* Method printPart print all part in the list
    * returns null
    */
   public void printPart()
   {
      System.out.println("Printing parts");
      for(int i=0; i<PartList.size(); i++)
      {
```

```
            part a =(part)PartList.elementAt(i);
            a.printData();
          }
    }

    /* Method getSize of part list
     * returns size of part list
     */
    public int getSize()
    {
       return PartList.size();
    }

   /* Method getPartAt requries the index as 'i'
    * returns part at position i in the list
    */
    public part getPartAt(int i)
    {
       return (part)PartList.elementAt(i);
    }


   /* Method assignParentID assigns each part a parentID by checking the
parent Name
    * returns void
    */
    public void assignParentID(AssemblyList asmlist)
    {
          for(int i=0; i<PartList.size(); i++)
          {
                   part a =(part)PartList.elementAt(i);
                   a.parentID = asmlist.getParent(a.parent);
          }
    }

    /* Method addID requires AssemblyList as 'asmlist' and a number as
'startID'
       * assigns each part in the list compID as the consequative
number
       * starting from startID returns the next highest number
       * used onlr for new MLCa database develoed by Bhagyashree
Suratran
     */
    public int addID(int  startID, AssemblyList asmlist )
    {
         for(int i=0; i<PartList.size(); i++, startID++)
         {
             part a =(part)PartList.elementAt(i);
             a.compID = startID;
               a.parentID = asmlist.getParent(a.parent);
         }
         return startID;
    }

}
```

```
/** Class XMLRead Implements a parser for reading the data from file
MLCA.XML
  * @Author - Bhagyashree Suratran
  * @Version - 1.0 of Nov 22, 1999
*/

import java.io.*;
import java.util.*;
import org.w3c.dom.*;
import com.sun.xml.tree.*;

class XMLRead
{
   XmlDocument xd;
   Element root;
   NodeList nl;
   Element parent;
   AssemblyList asmlist ;
   PartList prtlist;
   NodeList nlLocal, nlLocalChild;

   /* Method getAssemblyList
    * returns asmlist
    */
   AssemblyList getAssemblyList()
   {
       return asmlist;
   }

   /* Method getPartList
    * returns prtlist
    */
   PartList getPartList()
   {
       return prtlist;
   }

   /* Constructor for class XMLRead*/
   XMLRead()
   {
      try
      {
         Element local = null;
             asmlist = new AssemblyList();
             prtlist = new PartList();
         this.openXML("file:MLCA.xml");
         this.getRoot();
             this.nlLocal = this.root.getChildNodes();
             for(int i= 0; i<this.nlLocal.getLength(); i++)
             {

      if(this.nlLocal.item(i).getNodeType()==Node.ELEMENT_NODE)
                  {
                  local = (Element)this.nlLocal.item(i);
                          this.nlLocalChild =local.getChildNodes();
                          this.mapNodeData(this.nlLocalChild);
                  }
```

```
            }

        }
        catch(Exception e)
        {
            e.printStackTrace();
            System.out.println(e.getMessage());
        }
    }

    /* Method openXML requires a filename as string as 'xmlFile'
     * returns true if opened successfully else false
     */
    public  boolean openXML(String xmlFile)
    {
      try
      {
          xd = new XmlDocument();
          xd = XmlDocument.createXmlDocument(xmlFile, false);
          return true;
      }
      catch(Exception e)
      {
          System.out.println(e.getMessage());
          return false;
      }
    }

    /* Method getRoot to get root of the XML file
     *  returns void
     */
    public void getRoot()throws Exception
    {
      try
      {
          root = xd.getDocumentElement();
          System.out.println("In getRoot method : "+root.getTagName());
      }
       catch(Exception e)
       {
           System.out.println("Exception In method getRoot :
"+e.getMessage());
       }
    }

    /* Mathod mapNodeData requires a node as local and populated the
assemblyList or PartList
     * by checking the node whether it is a part data or assembly
     * returns void
     */
    public void mapNodeData(NodeList local)
    {
        String name, type, parent, material;
          double weight;
          Element local1;
          NamedNodeMap nnm;
        name=null;
```

```
        type=null;
        parent=null;
        material=null;
        weight = 0;
        int k=0;
        for(k= 0; k<local.getLength(); k++)
    {
            if(local.item(k).getNodeType()==Node.ELEMENT_NODE)
              {
                  local1 = (Element)local.item(k);
                    nnm=local1.getAttributes();
                    String data = "";
                    data = nnm.item(0).getNodeName();
                    String Value = "";
                    Value = nnm.item(0).getNodeValue();
                    if (data.trim().equalsIgnoreCase("type"))
                        type = Value;

                    else if (data.trim().equalsIgnoreCase("name"))
                        name = Value;

                    else if (data.trim().equalsIgnoreCase("parent"))
                        parent = Value;

                    else if (data.trim().equalsIgnoreCase("material"))
                        material = Value;

                    else if (data.trim().equalsIgnoreCase("weight"))
                        weight =(new Double(Value)).doubleValue();

                }

            }
        if(type.trim().equalsIgnoreCase("ASM"))
            {
               assembly asm = new assembly(name, parent);
               assembly checkasm = asmlist.isPresent(name);
           if(checkasm != null)
                  asmlist.updateQuantity(asm);
               else
                   this.asmlist.addAssembly(asm);
            }
            else if(type.trim().equalsIgnoreCase("PRT"))
            {
               part prt = new part(name, parent, material, weight);
               part checkprt = prtlist.isPresent(name);
               if(checkprt != null)
                    prtlist.updateQuantity(checkprt);
               else
                    this.prtlist.addPart(prt);
            }


}

/* Method assignID requres a starting number as 'ID'
 * returns void
```

```
     */
    public void assignID(int ID)
    {
        int nextID = asmlist.addID(ID);
        nextID = prtlist.addID(nextID, asmlist);
        asmlist.printAssembly();
          prtlist.printPart();

    }

}
```

```
/** Class assemblyList Implements a list for assembly
  * @Author - Bhagyashree Suratran
  * @Version - 1.0 of Nov 22, 1999
*/


import java.io.*;
import java.util.*;
import java.lang.*;
import java.sql.*;
import java.util.Properties.*;


public class MLCADriver
{
  AssemblyList asm;
  PartList prt;
  Connection c = null;
  String database, server, user, password;
  String driver;
  Statement st;
  XMLRead xr;

  /* Constructor for class MLCADriver */
  MLCADriver()
  {
    driver = "sun.jdbc.odbc.JdbcOdbcDriver";
    server = "jdbc:odbc:LCADB";
    user = "";
    password = "";
    xr = new XMLRead();
  }

  public static void main(String args[])
  {
    MLCADriver dv = new MLCADriver();
    dv.connect();
    //dv.check();
    //System.out.println("Material "+dv.getMaterialId("Plastic"));
    dv.populateData();
  }

   public void check()
   {
       prt =xr.getPartList();
     for(int i =0; i<prt.getSize(); i++)
       {
            part p = prt.getPartAt(i);
          System.out.println("for part "+p.name+" Material
"+p.material+" "+getMaterialId("Plastic"));
     }
   }
  /* Method connect for connecting to LCA database
   * returns void
   */
  void connect()
  {
```

```
    try
    {
       Class.forName(driver);
       c= DriverManager.getConnection(server, user, password);
       c.setAutoCommit(false);
       st = c.createStatement();
       DatabaseMetaData md = c.getMetaData();
       System.out.println("Connection Established");
    }
    catch(SQLException er)
    {
       System.out.println(er.toString());
    }
    catch(ClassNotFoundException ex)
    {
       System.out.println(ex.toString());
    }
 }

 /* Method populateData for populating the database
  * returns void;
  */
 void populateData()
 {
      asm = xr.getAssemblyList();
      prt = xr.getPartList();
      insertAssemblies();

 }

 void insertAssemblies()
 {
       int projectID = 0;
       String productName = null;
       try
       {
       for(int i =0; i<asm.getSize(); i++)
       {

              assembly a = asm.getAssemblyAt(i);

                  if (i == 0)
                  {
                        String qry = "INSERT INTO Project
(Project_Name) VALUES ('" +a.name+"')";
                        String projID = "SELECT Project_ID FROM
Project WHERE Project_Name = '"+a.name+"'";
                        st.executeUpdate(qry);
                        st.execute(projID);
                        ResultSet r = st.getResultSet();
                        Integer Proj_ID = new Integer("0");
                        while(r.next())
                        {
                              Proj_ID = new
Integer(r.getString("Project_ID"));
                        }
                        projectID = Proj_ID.intValue();
```

```
                        a.compID = projectID;
                        productName = a.name;


                    }
                    else
                    {
                        String productquery = null;

if(a.parent.trim().equalsIgnoreCase(productName.trim()))
                        {
                            productquery = "INSERT INTO Production
(Project_ID, Name, Root_ID, Identify_P2)"
                            +"
VALUES("+projectID+",'"+a.name+"','R', 'S')";

                        }
                        else
                        {
                            productquery = "INSERT INTO Production
(Project_ID, Name, Identify_P2)"
                            +" VALUES("+projectID+",'"+a.name+"',
'S')";
                    }
                    st.executeUpdate(productquery);
                    String PFS_ID = "SELECT PFS_ID FROM Production
WHERE Name = '"+a.name+"'";
                    st.execute(PFS_ID);
                      ResultSet r = st.getResultSet();
                      while(r.next())
                      {
                          a.compID = (new
Integer(r.getString("PFS_ID"))).intValue();
                      }

                }
               c.commit();

           }


      for(int i =0; i<prt.getSize(); i++)
    {
            part p = prt.getPartAt(i);
            String productquery = null;

      if(p.parent.trim().equalsIgnoreCase(productName.trim()))
                {
                    productquery = "INSERT INTO Production
(Project_ID, Name, Root_ID, Identify_P2)"
                        +" VALUES("+projectID+",'"+p.name+"','R',
'FP')";
                }
                else
               {
                        productquery = "INSERT INTO Production
(Project_ID, Name, Identify_P2)"
```

```
                        +" VALUES("+projectID+",'"+p.name+"', 'P')";
                }
                  st.executeUpdate(productquery);
                  String PFS_ID = "SELECT PFS_ID FROM Production WHERE
Name = '"+p.name+"'";
          st.execute(PFS_ID);
                  ResultSet r = st.getResultSet();
                  while(r.next())
                  {
                     p.compID = (new
Integer(r.getString("PFS_ID"))).intValue();
                }
                  c.commit();

      }
      asm.assignParentID();
      prt.assignParentID(asm);

    for (int i = 0; i<asm.getSize(); i++)
    {
            assembly a = asm.getAssemblyAt(i);
            if(a.parentID != (asm.getAssemblyAt(0)).compID)
            {
                  String relationQuery = "INSERT INTO
Relationship(Root, Child, Project_ID) VALUES("
                  +a.parentID+",
"+a.compID+","+(asm.getAssemblyAt(0)).compID+")";
                  st.executeUpdate(relationQuery);

            }
      }
    for(int i =1; i<asm.getSize(); i++)
    {
            assembly a = asm.getAssemblyAt(i);
            int parentID = a.compID;
            for(int j =0; j<prt.getSize(); j++)
            {
               part p = prt.getPartAt(j);
               if(p.parentID == parentID)
               {
                     String relationQuery = "INSERT INTO
Relationship(Root, Child, Project_ID) VALUES("
                     +parentID+",
"+p.compID+","+(asm.getAssemblyAt(0)).compID+")";
                   st.executeUpdate(relationQuery);
                     //System.out.println(parentID+" "+p.compID+"
"+(asm.getAssemblyAt(0)).compID);
               }
            }
      }
      //
      for(int i =0; i<prt.getSize(); i++)
      {
            String Name = null;
            try
            {
            part p = prt.getPartAt(i);
```

```
            Name = p.name;
            int materialID = getMaterialId(p.material.trim());
          String materialquery = "INSERT INTO PFS_Material (PFS_ID,
Material_Name, Weight,"
          +"Quantity, Project_ID, Unit,Material_ID)" +"
VALUES("+p.compID+",'"+p.material

+"','"+p.weight*1000+","+p.quantity+","+(asm.getAssemblyAt(0)).compID+
          ",'gm',"+materialID+")";

            st.executeUpdate(materialquery);
            String PFS_ID = "SELECT PFS_ID FROM Production WHERE Name =
'"+p.name+"'";
          st.execute(PFS_ID);
            ResultSet r = st.getResultSet();
            while(r.next())
            {
                  p.compID = (new
Integer(r.getString("PFS_ID"))).intValue();
            }
         }
         catch (SQLException e)
         {
              System.out.println("For part "+Name+" "+e.toString());
       }

      }
      c.commit();
      System.out.println("Insertion is Completed");
  }
  catch (SQLException e)
  {
      System.out.println("1");
        System.out.println(e.toString());
  }
  catch(Exception e)
  {
        System.out.println("2");
        System.out.println(e.toString());
  }

  }


  /* Method getMaterialId require a material Name as material
   * returns the materialID from database;
   */
  int getMaterialId(String material)
  {
        int flag = 0;
        try
        {
              String getMaxQuery ="Select Material_ID From Material
where Material_Name = '"+ material+"'";
              st.execute(getMaxQuery);
              ResultSet r = st.getResultSet();
              int MaxComp_ID = 0;
```

```
                while(r.next())
                {
                      flag = 1;
                    MaxComp_ID = r.getInt("Material_ID");
                }

           if (MaxComp_ID == 0)
           {
                      int ID = getMaxMaterialID() + 1;
                      String matInsert = "INSERT INTO
Material(Material_ID, Material_Name) VALUES ("+ID+",'"+material+"')";
                      st.executeUpdate(matInsert);
                      c.commit();
                      return getMaterialId(material);
                }

                return MaxComp_ID;
           }
           catch (SQLException e)
           {
                 System.out.println(material);
                 System.out.println(e.toString());
                 return -1;
           }
           //return -1;

   }

     int getMaxMaterialID ()
     {
         try
         {
               String getMaxQuery ="Select Material_ID from Material";
             st.execute(getMaxQuery);
             ResultSet r = st.getResultSet();
             int MaxComp_ID = 0;
             while(r.next())
             {
                MaxComp_ID = r.getInt("Material_ID");
             }
             return MaxComp_ID;
       }
       catch (SQLException e)
         {
            System.out.println(e.toString());
       }
       return 0;

     }


 }
```

APPENDIX E

SAMPLE OF MERX.XML FILE


```
<?xml version="1.0"?>
<start>
<Component> <Type type = " ASM" /> <Name name= " TELE99" /> <Parent
parent= "NULL" /></Component>

<Component> <Type type = " PRT " /> <Name name = "MAIN-HOUSING" />
<Parent parent= " TELE99" /> <Material material = " PLASTIC " />
<Weight weight=" 0.235585 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "BOTTOM-COVER" />
<Parent parent= " TELE99" /> <Material material = " PLASTIC " />
<Weight weight=" 0.105335 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "ROUND-BUTTON" />
<Parent parent= " TELE99" /> <Material material = " RUBBER " /> <Weight
weight=" 0.000541 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "ROUND-BUTTON" />
<Parent parent= " TELE99" /> <Material material = " RUBBER " /> <Weight
weight=" 0.000541 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "ROUND-BUTTON" />
<Parent parent= " TELE99" /> <Material material = " RUBBER " /> <Weight
weight=" 0.000541 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "ROUND-BUTTON" />
<Parent parent= " TELE99" /> <Material material = " RUBBER " /> <Weight
weight=" 0.000541 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "KEY-PAD-HOUSING" />
<Parent parent= " TELE99" /> <Material material = " RUBBER " /> <Weight
weight=" 0.033536 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "SPEED-DIAL-BUTTONS"
/> <Parent parent= " TELE99" /> <Material material = " PLASTIC " />
<Weight weight=" 0.010931 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "NUMBER-KEY-PAD" />
<Parent parent= " TELE99" /> <Material material = " PLASTIC " />
<Weight weight=" 0.021765 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "PIN" /> <Parent
parent= " TELE99" /> <Material material = " PLASTIC " /> <Weight
weight=" 0.000248 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "RUBBER-CUSHONING-
PART" /> <Parent parent= " TELE99" /> <Material material = " RUBBER "
/> <Weight weight=" 0.001103 " /> </Component>
```

```
<Component> <Type type = " PRT " /> <Name name = "H-SHAPE-PART" />
<Parent parent= " TELE99" /> <Material material = " PLASTIC " />
<Weight weight=" 0.000482 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "FLOATER" /> <Parent
parent= " TELE99" /> <Material material = " PLASTIC " /> <Weight
weight=" 0.005090 " /> </Component>
<Component> <Type type = " PRT " /> <Name name = "CLIP-SPRING" />
<Parent parent= " TELE99" /> <Material material = " STEEL " /> <Weight
weight=" 0.000437 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "UPPER-COVER" />
<Parent parent= " TELE99" /> <Material material = " PLASTIC " />
<Weight weight=" 0.202624 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "HAND-REST" /> <Parent
parent= " TELE99" /> <Material material = " PLASTIC " /> <Weight
weight=" 0.001260 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "MEMORY-PAD" />
<Parent parent= " TELE99" /> <Material material = " PAPER " /> <Weight
weight=" 0.000382 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "COVER-PLATE-OF-
MEMORY-PAD" /> <Parent parent= " TELE99" /> <Material material = "
PLASTIC " /> <Weight weight=" 0.001534 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "RECTANGULAR-PLATE" />
<Parent parent= " TELE99" /> <Material material = " PLASTIC " />
<Weight weight=" 0.014846 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "BASE-PLATE" />
<Parent parent= " TELE99" /> <Material material = " PLASTIC " />
<Weight weight=" 0.042547 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "BASE-PLATE" />
<Parent parent= " TELE99" /> <Material material = " PLASTIC " />
<Weight weight=" 0.042547 " /> </Component>

<Component> <Type type = " ASM " /> <Name name = "FUNCTION_KEYS" />
<Parent parent= " TELE99" /></Component>

<Component> <Type type = " PRT " /> <Name name = "FUNCTION-KEY-PAD" />
<Parent parent= " FUNCTION_KEYS" /> <Material material = " PLASTIC " />
<Weight weight=" 0.006952 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "OVAL-BUTTONS" />
<Parent parent= " FUNCTION_KEYS" /> <Material material = " PLASTIC " />
<Weight weight=" 0.000415 " /> </Component>

<Component> <Type type = " ASM " /> <Name name = "HANDSET" /> <Parent
parent= " TELE99" /></Component>

<Component> <Type type = " PRT " /> <Name name = "BOTTOMCOVER" />
<Parent parent= " HANDSET" /> <Material material = " PLASTIC " />
<Weight weight=" 0.069395 " /> </Component>
```

```
<Component> <Type type = " PRT " /> <Name name = "CUSHION" /> <Parent
parent= " HANDSET" /> <Material material = " FOAM " /> <Weight weight="
0.000362 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "RUBBER_INSERT" />
<Parent parent= " HANDSET" /> <Material material = " RUBBER " />
<Weight weight=" 0.001122 " /> </Component>
<Component> <Type type = " ASM " /> <Name name = "SPEAKER_UNIT" />
<Parent parent= " HANDSET" /></Component>

<Component> <Type type = " PRT " /> <Name name =
"SPEAKER_RUBBER_PROTECTION_" /> <Parent parent= " SPEAKER_UNIT" />
<Material material = " RUBBER " /> <Weight weight=" 0.004535 " />
</Component>

<Component> <Type type = " PRT " /> <Name name = "SPEAKER_" /> <Parent
parent= " SPEAKER_UNIT" /> <Material material = " RUBBER " /> <Weight
weight=" 0.034949 " /> </Component>

<Component> <Type type = " PRT " /> <Name name = "TOPCOVER11" />
<Parent parent= " HANDSET" /> <Material material = " PLASTIC " />
<Weight weight=" 0.047683 " /> </Component>

 </start>
```

REFERENCES

[1] Badwe, D., "Multi-Lifecycle Assessment of Cathode Ray Tubes" Masters Thesis for Industrial & Manufacturing Engineering Dept., New Jersey Institute of Technology, Newark, NJ, 1997.

[2] Boguski, Hunt, Cholakis, and Franklin, "LCA Methodology", *Environmental Lifecycle,* McGraw-Hill, New York, NY, 1996.

[3] Bylinsky, "Manufacturing for reuse", *Fortune Magazine*, Febraury 6, 1995.

[4] CATIA Solutions Magazine's Web site for CATIA available at http://www.catiasolutions.com/, March 99.

[5] IBM's Web site for CATIA, " IBM and Dassault Systemes Announce General Availability of CATIA Version 5 Release 3, Transforming Knowledge into Value, Automating Product Creation and Process Definition" available at http://www.catia.ibm.com/anncpres/anpro17.html, November, 1999.

[6] Parametric Technology Corporation's Web site for Pro/ENGINEER available at http://www.ptc.com/, June 99.

[7] Parametric Technology Corporation, "Conceptual Engineering White Paper", web site available at http://www.ptc.com/products/whitepapers.htm#cimdata_wp, August, 99.

[8] Structural Dynamics Research Corporation's Web site for I-DEAS Master Series available at http://www.sdrc.com/nav/software-services/ideas/, July, 99.

[9] Reggie Caudill, Donald Sebastian, and Bin Zhang , "Multi-Lifecycle Product Recovery for Electronic Products", *Multi-lifecycle Engineering and Manufacturing Program - Final Report 97,* Multi-Life Cycle Research Center, NJIT, Newark, NJ 1997.

[10] Reggie Caudill, "A Final Proposal to the National Science Foundation Engineering Research Centers Program", *Multi-lifecycle Engineering and Manufacturing Program Proposal*, Multi-Lifecycle Engineering Research Center, New Jersey Institute of Technology, Newark, NJ, September 1997.

[11] Caudill, Thomas, Kirchhoff, Kliokis and Linton "Lifecycle Analysis of CRTs", *Multi-lifecycle Engineering and Manufacturing Program - Final Report 99,* Multi-Lifecycle Engineering Research Center, New Jersey Institute of Technology, Newark, NJ, February 1999.

[12] Das, Sodhi, Caudill and Ji " Modeling and Estimating the Product Disassembly Effort", *Multi-lifecycle Engineering and Manufacturing Program - Final Report 98,* Multi-Lifecycle Engineering Research Center, New Jersey Institute of Technology, Newark, NJ, February 1998.

[13] Das, Sodhi and Caudill, "Demanufacturing and Disassembly Processes", *Multi-lifecycle Engineering and Manufacturing Program - Final Report 99,* Multi-Lifecycle Engineering Research Center, New Jersey Institute of Technology, Newark, NJ, February 1999.

[14] Elmasri and Navathe, "Database System Concepts and Architecture", *Fundamentals of Database Design,* Addison Wesley, Menlo Park, CA, 1994

[15] B.W.Vigon, "Life-Cycle Assessment: Inventory Guidelines and Principles", Risk Reduction Engineering Laboratory Office of Research and Development U.S. Environmental Protection Agency, February 1993.

[16] Joseph Fiksel "Conceptual Principles of DFE", *Design for Environment,* McGraw-Hill, New York, NY, 1996.

[17] Jane Ji, "Multi-lifecycle Assessment Design Tools and Software Development", Masters Thesis for Computer Engineering, New Jersey Institute of Technology, Newark, NJ, January 1999.

[18] Georg Lausen, and Gottfried Vossen., "Aspects of Object-Oriented Database", *Models and Languages of Object-Oriented Databases* Addison-Wesley, Menlo Park, CA, 1998.

[19] Hersh M., "A survey of Systems Approaches to Green Design with Illustrations from the Computer Industry", IEEE Transactions on systems, Man, and Cybernetics, Vol. 28, No.4, November 1998.

[20] Parametric Technology Corporations, *Pro/Toolkit Reference Guide.*

[21] Parametric Technology Corporations, *Pro/Toolkit User's Guide.*

[22] Saake, Conrad, Schmitt and Turker "Object Oriented Database design : what is the difference between relational database design?" paper presented in ObjectWorld, Frankfurt'95.

[23] SETAC Foundation, "A Technical Framework for Life-cycle Assessments" Pensacola, January 1991.

[24] Bengt Steen, Raul Carlson, Goran Lofgren, Goteborg "SPINE, A Relation Database Structure for Life Cycle Assessment", September 1995 The original report documenting the SPINE structure. Available at http://deville.tep.chalmers.se/SPINE_EIM /publicat.html, March, 99.

[25] Andrew Sweatman and Dr. Matthew Simon Presented "Design for environment tools and product innovation" at the 3rd International Seminar on Life cycle Engineering ECO-Performance 96, Zurich, Switzerland, March 18-20, 1996

[26] Jeffrey Ullman, "The Relational Data Model", *A First Course in Database Systems*, Prentice Hall, Columbus, OH, April 1997.

[27] John Wright and Larry Helsel, "The World of Materials", *Introduction to Materials and Processes,* Delmar Pub, Albany, NY, 1996.

[28] Hussan Fawzi Al-Okush,, "Design for Environment of office telephone and electronic products", Masters Thesis for Mechanical Engineering Dept., New Jersey Institute of Technology, Newark, NJ, January 1999.

[29] P. P. Chen "The Entity-Relationship Model – Towards a Unified View of Data", paper presented in ACM Transactions on Database Systems, March 1976

[30] Ali Dogru, Murat Tanik, Donald Sebastian, Sanchoy Das, "Tool Kit of Existing Best Practices for Computer & CRT Demanufacturing and DFE", *Multi-lifecycle Engineering and Manufacturing Program - Final Report 98,* Multi-Lifecycle Engineering Research Center, New Jersey Institute of Technology, Newark, NJ, February 1998.

[31] Bruce W. Vaigon in "Software Systems and databases", *Environmental Lifecycle Assessment,* McGraw-Hill, New York, NY, 1996