The Component-oriented Approach towards Complex Product Development

Fujun Wang, John J. Mills
Automation & Robotics Research Institute
The University of Texas at Arlington, Fort Worth, TX76118
{fwang, jmills}@arrirs04.uta.edu

Keywords Virtual Product Development, Collaborative Product Development, Software Component, System Architecture, Product Data Modelling

Abstract This paper proposes a new approach towards complex product development. The approach is based on the software component mechanism. A software component is a reusable software package which can run across heterogeneous platforms. This paper presents a Component-based Open System Architecture, in which the design of a product component is implemented into a software component. The product component data and operation methods are exposed and can be reused by others. The Collaborative Product Representation Model is proposed to represent the product component in this environment. It is a neutral, understandable, customizable and reusable model with self-management capability. The Product Model Processor helps to define this complex model data.

1. INTRODUCTION

This paper presents an approach towards the development of complex products. A complex product refers to one with more than one components, which is usually designed by one more persons or organizations. An airplane is a complex product. A gear box can also be looked as a complex product if its design needs more than one person. How to develop such a complex product in short time with reliable quality is always the pursued aim. The World-wide Virtual Product Development is an important methodology for the emerging global marketplace [11,12,20,21,22]. With this development mode, a complex product is developed collaboratively and virtually by a temporary alliance with world-wide scope. Product components developed by different alliance members in different locations are assembled into the

J. P. T. Mo et al. (eds.), *Global Engineering, Manufacturing and Enterprise Networks*

final product at another location. Both the enterprise-wide collaboration and the world-wide development have the same development feature, i.e., the product components are developed by different persons or organizations in a distributed environment, and then they are assembled into the final product. This paper proposes an approach supporting this development feature. The key issue here is how to represent the product component and manage them to make them easily accessible and usable in a network environment. The requirements and previous research are outlines below.

- *Open System Architecture.* Since the product development alliance is possibly dynamic in the virtual product development mode, an open distributed system is required. Gauchel et. al. [8] proposed the concept of *Autonomy* and used it in product modelling. Cutkosky [5] proposed an agent based design system running on the Internet. Through a resource directory on the Internet, those agents can be found and then be accessed.
- *Customizability.* A customizable product component model can always provide much more freedom for reuse by others. Most of the research in this field is in three areas: the *Parametric Model*, the *Constraints-based Model* and the *Serialized Model.* The Parametric Model is a variables-based product data representation. Dai [6] presented a parametric feature-based model for integrated CAD/CAPP/CAM systems in his Ph.D thesis. The Constraints-based Model is always coupled with the Parametric Model. It uses the constraints to limit the scope of a variable and the relationships among one or more variables. Eastman [7] embedded integrity rules that apply between applications into his model. The Serialized Model refers to the representation of serial products or product families. Gero and Shi [10] built a product families model based on the phase transition in the biological world.
- *System Methods Reuse.* This might be the most important need and is different from other traditional models. In a traditional collaborative design session, what is exchanged between designers is the product model data and design knowledge. However, the system methods which operate on the product data can not be shared. In fact, system methods are an important design resource.
- *Neutral Model.* Since different product components are designed by different departments or persons, it is necessary to define those product components in some neutral model to promote their data exchange smoothly. One accepted approach to this problem is the Feature-based Modelling basing on some product data exchange standard as STEP [15,16].

- *Understandable Model.* The feature-based data is limited to geometry data and a few engineering properties, which is far away from the requirement of information exchanged between partners. Therefore, knowledge is often embodied in a product model called an *Ontology* [1]. Gero and Rosenman proposed the paradigm of '*purpose-function-behaviour-structure*' which describes the product engineering semantics [9,23,24,25]. Henderson [13] represented functionality and design intent in a meta-model. In addition, design rationale is captured in some systems to explain the design principle used [2,17].
- *Self-Management.* Since a product component can be serialized and parameterized, it is necessary to improve its robustness. This can be realized by enabling the product component to manage itself automatically. *Agent-based Management* is an AI approach to maintaining the product data automatically. Rosenman and Wang [24] proposed a component agent model for the representation and management of a product object.

2. COMPONENT-BASED OPEN SYSTEM ARCHITECTURE

2.1 Component-oriented Approach

As an approach to software development, *Object-Oriented* (OO) Analysis and Programming has been in a dominant position for one to two decades. Its approach is that of encapsulating attributes and methods in a class and permitting inheritance between classes (fig.1). Code reuse is available by the inheritance mechanism, but this reuse is limited inside an application, or on the class-level. Such a general application has two main parts, the Implementation and the Data Base (fig.1a). Communication between two applications is through the Data Base. That is, the interaction is limited to sharing data only. Therefore, the code reuse or system function sharing between two applications is impossible. However, the main task of distributed systems integration is the inter-operation between them: the code reuse.

Though the OO approach brought a giant revolution from traditional software development, the promise of large scale of code reuse did not become reality [26]. Recently, a new approach, *Component-Oriented* (CO) approach is becoming the focus in software industry. A component is a reusable software package. A *Component Based Application* (CBA) provides services containing the data operations and method operations in its implementation. They are the reusable attributes and methods which are

shown in fig.1b by the small solid circles and boxes. An Interface File coupled with these services describes the exposed data and methods. If a CBA works as a server, the Interface File is the medium connecting the server and one or more clients. A traditional application can be wrapped into a component to expose its data operation [28] (fig.1c).

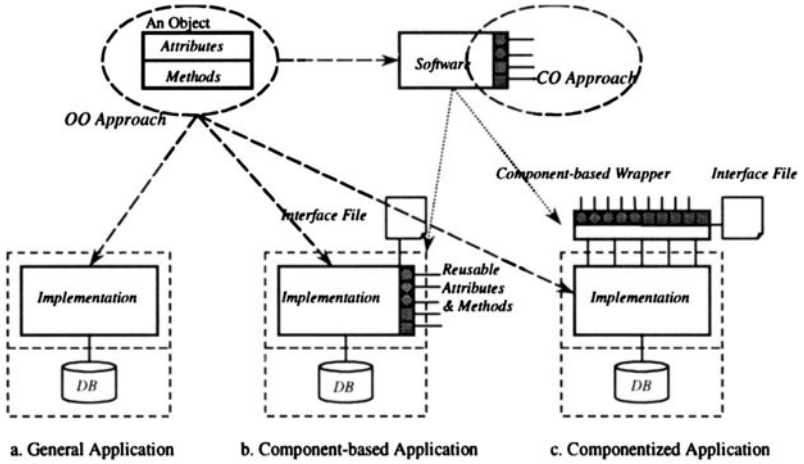

Figure 1. OO Approach and CO Approach

2.2 Component-based Open System Architecture

The reuse-ability of a component allows it to work as a *System Component*, or as part of a large system. CORBA, COM and EJB are sometime called distributed object 'middleware', because they mediate between components to allow them to work together, integrating them into a single, functional whole [14]. A distributed system made up of dozens of component-based applications can be called a *Component-based System.* Fig.2 indicates such a system. These System Components can be distributed in heterogeneous platforms. Each system component implements the design, analysis or manufacturing of a *Product Component* and supports to assemble high level product component or the final product. A system component might encapsulate some legacy application such as the CAD, CAPP, CAM, or some proprietary system. Because those system components are distributed, a central management facility—the *Web-based Design Resource Manager* manages the registering and accessing of those system components. This system is called an open system because of the distribution, independence and inter-operation of the component mechanism. The system component is not pre-defined and the number of the system components is not fixed. We call the system architecture the *Component-based Open System Architecture* (COSA).

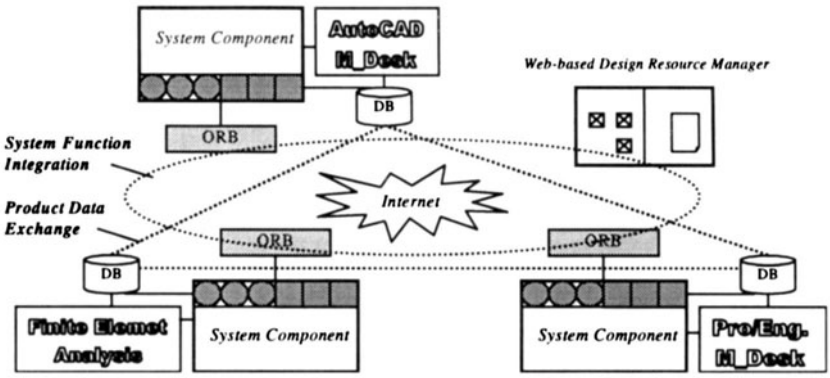

Figure 2. Component-based Open System Architecture

3. PRODUCT COMPONENT DESIGN

3.1 Product Component Model

The section covers the representation of a Product Component in the Component-based Open System Architecture (COSA). This paper presents a richer model to meet the five needs noted in section 1. The model here is called *Collaboration-oriented Product Representation Model* (CPRM). It is the combination of four data modules: *Functional Data*, *Embodiment Data*, *Management Data* and *Reusable Data*. Fig.3 indicates the model architecture. It should be noted that the Reusable Data is added and the Structural Data is replaced by the Embodiment Data. The Reusable Data represents the feature of a software component, i.e., the reusable attributes and methods. The Embodiment Data contains more semantics than Structural Data.

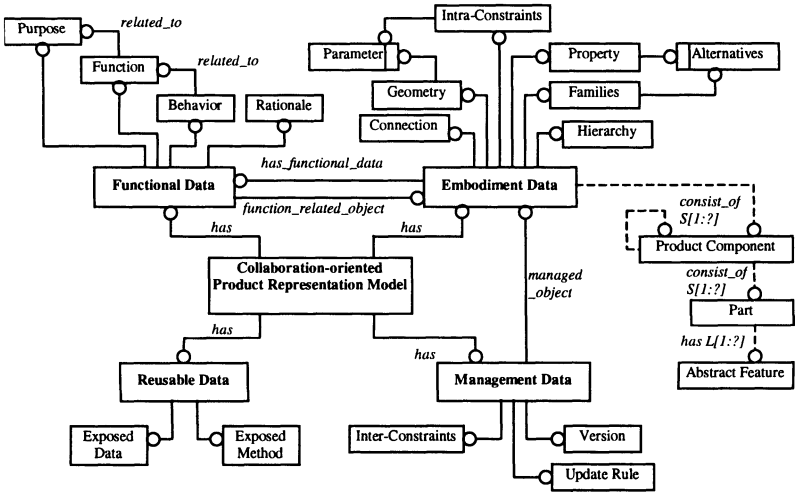

Figure.3 Collaboration-oriented Product Representation Model

Figure.3 Collaboration-oriented Product Representation Model

1). Functional Data

Functional Data describes the information related to what a product object is for, what it does and why it is what it is. It is decomposed into four types of UoF: *Purpose, Function, Behavior* and *Rationale*. The concept of UoF is from STEP [15,16], which is used to define the data unit with single function [23,24].

- *Purpose* is essentially the requirements for the products and describes the needs of clients and intentions of designers. It is the pursued aim of the design activity. Generally, it is more conceptual.
- *Function* describes what the product object does. Intended functions are those which are formulated as necessary to fulfill the purpose. Unintended functions may occur and it is important to note these too.
- *Behavior* illustrates the working principles of the object and logical actions or influence on other objects. The required behaviors are those which are necessary to produce the required or intended functions. The relationship between purpose, function, behavior and embodiment can be summarized as: PURPOSE enabled by FUNCTION achieved by BEHAVIOUR exhibited by EMBODIMENT [23,24]. Note that we have replace Rosenman and Gero's idea of "Structure" with "Embodiment", which is a term better suited to mechanical design.
- *Rationale* contains the reasons or justification of design decisions in terms of selecting values for structure variables to satisfy behavior constraints or values.

2). Embodiment Data

Embodiment Data describes the product structural or physical information, the core data module of CPRM. This data module is neutral and customizable. It has 6 types of UoF (*Unit of Functionality*): *Families, Hierarchy, Connection, Geometry, Property,* and Intra-*Constraints* (fig.3).

- *Families* represent the classes or types of an abstract feature, a product part, a product component or the whole product (called Product Object). For example, the turbine engine has subclasses of turbo-fan, turbo-jet, turbo-shaft and turbo-propeller, and each subclass engine has several types, even a special type has serial engines [4]. The CFM56 is a type of turbo-fan engine, but it has serial types as the CFM56-2, -3, -5A, -5B etc. [27]. The families relationship is a class hierarchy. It can be represented by "*Is_Type_Of*" or "*Has_Types*".
- *Hierarchy* describes the structural hierarchy of a product object, which is different from the *Families*. For example, a common turbo-fan engine is made up of fan, compressor, combustor, turbine and accessories. Each

of these component is also made up of sub-components. This relationship can be represented by "*Is_Component_Of*" or "*Has_Components*".

- *Connect* indicates the object relationship in a level of the product object hierarchy. For example, the fan is connected to the compressor. This relationship can be represented by "*Connected_To*".
- *Geometry* represents the geometry shape, dimensions and location of a product object. We use the application protocol STEP 203 [15] as our geometry standard. The geometry shape has many types and "*Alternatives*" can be listed for designers. In order to realize a parametric model, the dimensions can be represented by 'Variables', and the location should also be variable or relative to its related objects
- *Property* represents the engineering attributes of a product object such as the material and processing attributes. The application protocols STEP 203 and 214 [15,16] are used as our standards for this representation. To make the property customizable, it should be represented by "*Alternatives*", i.e., alternative properties.
- *Intra-Constraints* represents the constraint relationship inside a product object. The constraints here are limited to the geometry parameters. The parametric data with constraints realizes a real parametric object model. The constraints among different product objects are called *Inter-Constraints* which are classified into the Management Data.

3). Management Data

The information for maintaining and controlling the model data in the design process is classified as management data. It contains three types of UoF: *Inter-Constraints*, *Update Rule* and *Version.*

- *Inter-Constraints* define the geometry parameter constraints among multiple product objects in a product. They are in the form of formula, ranges, or rules. Inter-Constraints play an important role towards a parametric product model.
- *Update Rule* represents the action behavior on the change of model data. It is coupled with the data as the parameter constraints, product families, hierarchy etc., and it operates as a watch dog for those data. There are two types of update rules: *Condition/Action* and *Action/Condition/Correction* [18].
- *Version* serves to capture the product data evolutionary history, including not only the version number, but also the related designer, and updated time.

4). Reusable Data

The Reusable Data contains the UoFs of *Exposed Data* and *Exposed Methods* (fig.3).

- *Exposed Data* represents the Data Service which is realized by methods '*GetParameter()*' and '*SetParameter(...)*'. The '*GetParameter()*' gets the data value and the '*SetParameter(...)*' sets the data value. For example, if you want to get or set the length value of some object, the methods '*float L=GetLength()*' and '*void SetLength(L)*' can be used.
- *Exposed Method* represents the Method Service which exposes the system implementation methods which contain the data base accessing, performance calculation, constraints management and so on.

3.2. Product Model Processor

Section 3.1 proposes a complex product model with four data modules which are made up of fifteen UoFs and multiple relationships. Inputting these data becomes a major concern, especially when the geometry is perhaps entered using a commercial CAD System. To facilitate data entry and to help with converting of CAD system derived geometry, we are suggesting the concept of the Product Model Processor which always works with a CAD system. Generally, a designer designs the product object in a CAD system. The model is usually feature-based and the data is limited to the geometry data with some engineering properties. This data is called the Original Data. Then the original data is processed by the PMP. This process is in 5 steps. (I) Transferring the original data into the neutral data which is Abstract Feature-based, (II) Encapsulating more engineering semantics onto the neutral data towards to the Embodiment Data, (III) Customizing the model by parameterizing the geometry data, engaging constraints on them and serializing the model, (IV) Encapsulating Functional Data and Management Data onto the Model, (V) Activating the Enabling Service which exposes the model data and methods. This process makes it easy to define the complex CPRM data for a product object. To provide this process, the PMP has the following modules (shown in fig.4).

- **Model Broker**

The Model Broker transfers the model data between the special model data of any CAD system and the abstract feature based neutral data of CPRM. Since most of the data models of current CAD systems are feature-based, the model translation in the Model Broker is the feature.

- **Neutral Model Definer & Browser**

As the name shows, this system module has two functions: serving as a data browser and definer. The data coming from the Model Broker can be viewed in the browser. The definer is used to extend the data from the Model Broker or to define a product object into the neutral model. The definer contains a *Product Composition Definer* and dozens of *Abstract Feature Units*. The Product Composition Definer is used to define the composition of a PC

including its sub-components, parts and their hierarchy or connection relationship. The Abstract Feature Unit allows users to define the neutral feature data which is independent of any shape representation required by a CAD application.

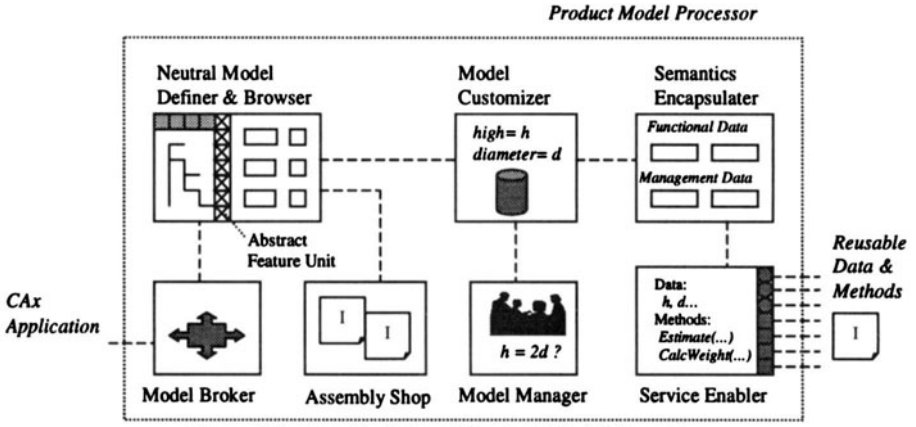

Figure.4 System Modules of Product Model Processor

- **Model Customizer**

The model data in the Neutral Model Definer & Browser is usually specific with specific values. This Model Customizer module is for transferring this special data into a class data which is customizable. This customizing process is in 3 steps. (I) *Parameterizing* the geometry data and extending the engineering properties. The special geometry data value is replaced by the related variable or parameter. The special engineering property as the material can be extended by alternatives, (II) Merging *Constraints* on those parameters. This work defines the limitation of a parameter and the constraint relationship among one more parameters, which enables the whole model become a parametric one, (III) *Serializing* the product families. It summarizes and classifies the product families. A special product in the product families is featured by its special function, structure and engineering performance, which is embodied by its components, parts even features. Therefore, the work here is not only to classify the product families and represent their featured data, but also to classify their components (parts, features) families and their distinguished data.

- **Semantics Encapsulator**

The Semantics Encapsulator helps a designer to define the Functional Data and some of Management Data, and combines to the Embodiment Data. The Functional Data contains the design purpose, object function & behavior, and decision rationale. The constraints have been defined in the Model

Customizer. The update rules can be attached to those constraints. The version of the model data can be updated automatically in the design process.

- **Service Enabler**

The Service Enabler serves to expose the model data & methods, i.e., produces the Reusable Data. The exposing process is a programming process usually implemented in the system development environment such as the Boland's JBuilder enterprise edition [3] and the Microsoft's Visual C++ enterprise edition [19] which supports the CORBA or COM based application development. To make it convenient for a user to expose the reusable data, a simple Application Programming Interface (API) is provided to perform the task. This API is called the Service Enabler.

- **Assembly Shop**

The Assembly Shop (AS) is an Application Programming Interface (API) environment for a designer to integrate the system function of related System Components (SC) which can be found from the Web-based Design Resource Manager. In this AS the reusable attributes and methods of related product components implemented in other system components can be reused if the designer wants to assemble those product components.

- **Model Manager**

The Model Manager maintains the model data consistency containing the product families, hierarchy, constraints, version and functional relationship. The product family management manages the creation of new product type, and the consistency between the product type and its special featured data. The hierarchy management manages the product composition. The constraints management manages the parameter constraints and update rules to keep the consistency of the model data. The version management captures and maintains the product design evolution history. The functional relationship management maintains the relationship between the purpose, function and product structure. The detail approaches to those management facilities will be described elsewhere.

4. CONCLUSION

A complex product is made up of multiple product components which are possibly designed by different persons or organizations at distributed locations. This paper proposes an approach towards the complex product development. This approach is based on the Component technology. A Component is a reusable software package. The design of each product component is encapsulated into such a (software) Component. The model data and operation methods of the product component can thus be reused by

others because of the Component mechanism. The design of the product component is in a network environment implemented by a *Component-based Open System Architecture* (COSA) where a *Web-based Design Resource Manager* is responsible of managing those distributed components. In such an environment, the complex product's components can be designed in different places and then be assembled together. Word 'Assemble' refers not only to the assembly of product component data, but also that of the data operation methods.

In this COSA environment, the representation of the product component is another important issue. This paper presents a *Collaborative Product Representation Model* (CPRM) which is neutral, understandable, customizable and reusable model with self-management capability. It is the combination of *Functional Data*, *Embodiment Data*, *Management Data* and *Reusable Data*. The CPRM is a complex product data model with a few of units of functionality. In order to simply the definition of such a complex model data, this paper presents a *Product Model Processor* (PMP) helping a user to define the model data step by step or helps to wrap a special CAD model data into the CPRM data. The PMP is made up of six modules, a *Model Broker*, a *Neutral Model Definer & Browser*, a *Semantic Encapsulator*, a *Model Customizer*, a *Service Enabler*, an *Assembly Shop* and a *Model Manager*.

We believe that this approach is not only effective for product distributed development, but also valuable for the product data management in an enterprise. The reusability and customizability of the product model are valuable for the evolved product development. It will save the development cost and shorten the time to market.

REFERENCES

[1] Alberts, L. K., "YMIR: A Sharable Ontology For The Formal Representation of Engineering-Design Knowledge", see at http://wwwis.cs.utwente.nl.8080/kbs/publications/ontology-pubs.html, 1994.

[2] Ball, N., Matthews, P. and Wallace, K., "Managing Conceptual Design Objects", in J. S. Gero and F. Sudweeks (eds), *Artificial Intelligence in Design'98*, 1998, pp.67-86.

[3] Borland Corp., "JBuilder 3 Enterprise Developer Guide", 1999.

[4] Chen, G., Ma, M. and Xiao, L., "Structural Design of Gas Turbine Engine", Published by Beijing University of Aero. & Astro, 1988.

[5] Cutkosky, M.R., "Agent-based Concurrent Design", Advances in Concurrent Engineering, Aug.26-28, Toronto, Canada

[6] Dai, S., "Research on the STEP-based Integrated Product Model and the Implementation of an Integrated Design Platform", *Ph.D Thesis*, Beijing University of Aeronautics & Astronautics, 1997.

[7] Eastman, C. M., "Managing Integrity in Design Information Flows", *Computer-Aided Design*, Vol.28, No.6/7, 1996, pp.551-565.

[8] Gauchel, J., Vanwyk, S., Bhat, R. R. and Hovestadt, L., "Building Modelling Based on Concepts of Autonomy", in J. S. Gero (eds), *Artificial Intelligence in Design'92*, 1992, pp.181-197.
[9] Gero, J., "Design Prototypes: A Knowledge Representation Schema for Design", *AI Magazine*, Winter, 1990, pp26-36.
[10] Gero, J. and Shi, X.G. Design development based on an analogy with developmental biology, *CAADIA'* 1999. *Proceedings of the Fourth Conference on Computer Aided Architectural Design Research in Asia*, Shanghai, P.R.China. pp.253-264.
[11] Goldman, S., , Nagel, R.N., and Preiss, K., "Agile Competitors and Virtual Organizations", Pub. By Van Norstrand Reinhold, New York, 1995.
[12] Hart, D., Patterson, R, and Neal, R, "Next Generation Manufacturing: A Framework for Action", Executive Overview, The Agility Forum, Bethlehem, PA, January, 1998.
[13] Henderson, M. R., "Representing Functionality and Design Intent in Product Models", in 2nd ACM Solid Modelling'93, Montreal, Canada.
[14] Inprise Corp., "Distributed Object Computing in the Internet Age", see at http://www.inprise.com/visibroker/papers/distributed/wp.html, 1999.
[15] ISO/STEP, "10303-203: Product Data Representation and Exchange--Part 203: Application Protocol: Configuration Controlled Design", 1996 (a).
[16] ISO/STEP, "10303-214: Product Data Representation and Exchange--Part 214: Application Protocol: Core Data for Automotive Mechanical Design Processes", 1996 (b).
[17] Klein, M., "Integrated Coordination in Cooperative Design", *International Journal of Production Economics*, 1995.
[18] Mackellar, B. K. and Peckham, J., "Multiple Perspectives of Design Objects", in J. S. Gero and F. Sudweeks (eds), *Artificial Intelligence in Design'98*, 1998, pp.87-106.
[19] Microsoft Inc., "COM Home", at http://www.microsoft.com/com, 1998.
[20] Mills, J.J., "Information Infrastructures: Facilitating Global Design and Production", *9th International Conference on Production Technology (PTK 98)*, Berlin, Germany, October 29-30, 1998a.
[21] Mills, J., "Introduction: Towards an Information Infrastructure for Manufacturing Industry", in John J. Mills and Fumihiko Kimura (eds), *Information Infrastructure Systems for Manufacturing II*, 1998b.
[22] Mills, J., Brand, M. and Elmasri, "AeroWEB: An Information Infrastructure for the Supply Chain", in John J. Mills and Fumihiko Kimura (eds), *Information Infrastructure Systems for Manufacturing II*, 1998c, pp323-336.
[23] Rosenman, M. A. and Gero, J. S., "Modelling Multiple Views of Design Objects in a Collaborative CAD Environment", *Computer-Aided Design*, 1996, Vol.28, No.3, pp.193-205.
[24] Rosenman, M. A. and Wang, F. J., "CADOM: a Component Agent Model based Design-Oriented Model for Collaborative Design", Submits for *Research in Engineering Design*, 1999a.
[25] Rosenman, M. A. and Wang, F. J., "A Component Agent Based Open CAD System for Collaborative Design", Submitted for *Automation in Construction*, 1999b.
[26] Sterling Software, "Component Based Development and Object Modelling", at http://www.cool.sterling.com/cbd/whitepaper/, 1998.
[27] Wang, F. J. and Chen, G., "The Design Evolution of CFM56 Families", in Preceding of Structure, Strength and Vibration of Aero-Engine, China, Oct., 1994.
[28] Wilkes, L., "Legacy Componentization and Wrapping", *Component Strategies*, 1(8), Feb'99, p50-57.