

**DESIGN SYNCHRONIZATION IN DISTRIBUTED
COLLABORATIVE DESIGN – DESIGN CHANGE IN
PRODUCT-PROCESS DESIGN ACROSS GLOBAL
ENTERPRISES**

Bok Shung Hwee

(B. ENG (HONS), M. Eng)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF MECHANICAL ENGINEERING

NATIONAL UNIVERSITY OF SINGAPORE

2007

The Water Is Wide

ACKNOWLEDGEMENTS

I would like to sincerely thank my advisors for their support and guidance. I am grateful to Professor Andrew Nee for his mentorship since 1987, personal words of significance and encouragement, and professional research leadership and advice. To Professor Wong Yoke San, I am appreciative of his support in accepting me into LCEL and facilitating research activities. To Associate Professor Senthil Kumar, thank you for your support too.

My journey in pursuing this PhD may at last end with but a small contribution in knowledge, and may there be a new hope and chapter ahead.

To God Be the Glory.

TABLE OF CONTENTS

Acknowledgements	ii
Table of Contents	iii
List of Figures	vii
List of Tables	xi
Summary	xii
CHAPTER 1 INTRODUCTION	1
1.1 Background	1
1.2 Motivation and Purpose	3
1.3 Organisation of the Thesis	6
CHAPTER 2 LITERATURE REVIEW	8
2.1 CIM, CE, CAPP and GLOBAL MANUFACTURING	8
2.2 Digital Enterprise Technology (DET) Cornerstones	12
2.3 Digital Enterprise Technology (DET) Functionality Issues	17
2.4 Related Work	19
2.5 Problem Statement and Research Objectives	31
CHAPTER 3 MIDDLEWARE FRAMEWORK AND APPLICATION	38
ARCHITECTURE FOR DISTRIBUTED	
COLLABORATIVE DESIGN	
3.1 Conventional CAD Systems	39

3.2 Middleware Framework and Architectural Elements	44
3.2.1 Classification and Distribution of Functionality and Data	45
3.3 Applications Architecture and Computing Environment	48
3.3.1 Distributed Client-Server Architecture	49
3.3.2 Geometric Modelling Server	55
3.3.3 Product Model and Data Representation	56
3.3.4 Application View	57
3.3.5 Reusable client classes for application views	58
3.4 Distributed Collaborative Design and Design Synchronization	59
3.5 Discussion and Summary	65
CHAPTER 4 FRAMEWORK DEVELOPMENT AND INTERACTIVE	68
FIXTURE DESIGN APPLICATION IN DISTRIBUTED	
COLLABORATIVE DESIGN	
4.1 System Architecture and Overview	68
4.2 Application View	70
4.2.1 Visualization	71
4.2.2 Client Infrastructure	74
4.2.3 Application View Visualization Functionality	76
4.3 Server Infrastructure and Geometric Modelling Services	78
4.3.1 Server Infrastructure	79
4.3.2 Geometric Modelling Services	80
4.3.3 Modelling Interface and Functions	83
4.3.4 Product Modelling Server Architecture	86
4.4 Product Modelling With XML	89

4.5 Interactive Fixture Design Application	93
4.5.1 Fixture Design Methodology and Application Architecture	93
4.5.2 Design Synchronization with Interactive Fixture Design	102
4.6 Discussion and Summary	103
CHAPTER 5 DESIGN SYNCHRONIZATION MIDDLEWARE	107
MECHANISMS FOR EFFECTIVE DESIGN CHANGE	
UPDATE	
5.1 Design Synchronization Considerations for Application View Updates	108
5.1.1 Interactive Visualization in Distributed Collaborative Design	109
5.1.2 Graphics Simplification Techniques	111
5.1.3 Graphics Compression Algorithms	114
5.2 Leveraging Model Compression for Design Synchronization	116
5.2.1 Model Compression Algorithm	116
5.2.2 Product Modelling Architecture with Integrated Model Compression	120
5.2.3 Augmented Product Data Representation	122
5.3 Experimental Results of Integrated Model Compression	125
5.4 Design Synchronization for Design Change	128
5.5 Local Face Model Compression for Design Change Synchronization	129
5.6 Design Change Detection within Shape Modification	131
5.7 Boundary Representation Model Changes	133
5.8 Boundary Representation-Based Design Change Detection	138
5.9 Design Change Synchronization for Application View Update	142
5.10 Discussion and Summary	144

CHAPTER 6 DESIGN SYNCHRONIZATION FOR COLLABORATIVE	147
DECISION MAKING	
6.1 Introduction	147
6.2 Design Change Detection and Update	148
6.3 Design Change Synchronization Case Study with Fixture Design	151
6.4 Design Synchronization with Application Relations Management	158
6.5 Summary	162
CHAPTER 7 CONCLUSIONS AND RECOMMENDATIONS	164
REFERENCES	168
PUBLICATIONS ARISING FROM THIS THESIS	176

LIST OF FIGURES

Figure 2.1: DET Theoretical Cornerstones	14
Figure 2.2: Importance of Early Conceptual Design Decisions	14
Figure 2.3: Availability of Design Tools	15
Figure 2.4: Master model architecture with client views	25
Figure 3.1: Distributed Industrial Environments - Vertical to Horizontal Fragmented Value Chains	44
Figure 3.2: Distribution of Functionality and Data - 1.) Distributed Design Changes; 2.) Product Model Components; & 3.) Requirements and Considerations	45
Figure 3.3: Proposed Application Architecture based on Master Modellers and Client Application Views	51
Figure 3.4: Middleware Framework – A Layered Perspective	53
Figure 3.5: Product Modeling in Distributed Environments - Application Views & Relationships with Relevant Design Synchronization Support for the Example of a Forged Car Rim	60
Figure 3.6: Product Modeler Architecture	63
Figure 3.7: Workpiece Design and Corresponding Fixture Design	65
Figure 3.8: Design Application View	65
Figure 4.1: System Architecture for Interactive Fixture Design	69
Figure 4.2: A Shape3D Visual Object(s) inside a Java3D Scene Graph	71
Figure 4.3: Symbols Used in Representing Java3D Scene Graph	72
Figure 4.4: A Java3D Scene Graph Integrating Scene Graph's Object Space with a View/Screen Canvas	72
Figure 4.5: Rendering Object Space on Image Plane in a Virtual Universe	72
Figure 4.6: Application View with Java3D Canvas for Interactive Fixture Design	74
Figure 4.7: Class Architecture on Client Side	75
Figure 4.8: Class Architecture on Server End	82

Figure 4.9: Block Represented By Its Boundary	85
Figure 4.10: Example of a Tessellated Model	85
Figure 4.11: Basic Product Modeling Server Architecture	88
Figure 4.12: DTD Schema of Product data XML file	90
Figure 4.13: Actual DTD of the XML file of Geometric Data of a Body	90
Figure 4.14: An Illustration of the Product Data XML	92
Figure 4.15: Interactive Fixture Design Sequence	95
Figure 4.16: Workpiece and Corresponding Fixture Design	95
Figure 4.17: Example of a hole-based fixture base plate	96
Figure 4.18 Example information stored in the fixture element database	96
Figure 4.19: Support Rule Implementation and View Interaction	100
Figure 4.20: Locator Rule Implementation and View Interaction	101
Figure 5.1: Classification of 3D Models – Geometric Complexity vs Combinatorial Complexity	110
Figure 5.2: CLERS Illustration	118
Figure 5.3: Model Compression Traversal	119
Figure 5.4: Model Compression and Decompression Procedures	120
Figure 5.5: Basic integration and sequence of creating the augmented Product Data schema	121
Figure 5.6: Product Modeler Architecture with Model Compression and Design Change Detection	121
Figure 5.7: Augmented Product Data schema incorporating compressed geometry	123
Figure 5.8: Illustration of Augmented Product Data schema	124
Figure 5.9: A Chuck Workpiece	126
Figure 5.10: A Flange-like Workpiece	126
Figure 5.11: Additional Results of Integrated Model Compression	127

Figure 5.12: An interactive demonstration of face selection for compression	130
Figure 5.13: Corresponding face compression results	130
Figure 5.14: Interactive fillet modeling operation with compression of selected generated face	131
Figure 5.15: Compression of face mesh corresponding to fillet operation	131
Figure 5.16: Boundary Representation Graph Model	134
Figure 5.17: Illustration of Types of Topological Shape Changes	137
Figure 5.18: Illustration of B-rep face shape entity state changes	137
Figure 5.19: Illustration of B-rep shape entity operations inside design change	138
Figure 5.20: Sequence of steps to carry out shape modification with change detection	139
Figure 5.21: Design Change Detection Algorithm for Face Shape Entity	140
Figure 5.22: Improved Augmented Product data schema to support design change	143
Figure 5.23: The filleted block with new and replaced face shape entities	143
Figure 6.1: An Arm Case Workpiece	149
Figure 6.2: Highlighted Affected Faces in old B-rep Model before Design Change	150
Figure 6.3: Modified/Replaced and New Faces Detected in Design Change	150
Figure 6.4: Modified/Replaced, New and Mapped Faces in new B-rep Model after Design Change	151
Figure 6.5: Workpiece before Design Change in Product Design Application View	152
Figure 6.6: Typical Output of Model Compression of Workpiece	153
Figure 6.7: An Initial Fixture Configuration in Application View before Design Change	154
Figure 6.8: Workpiece after Design Change in the Product Design Application View	154

Figure 6.9: Captured Face Shape Entities in Design Change Detection	154
Figure 6.10: Captured Face Shape Entities for Design Change Update through Compression	155
Figure 6.11: Fixture Design with Design Change Update on Application View	156
Figure 6.12: Fixture Re-Design Completed on Application View	156
Figure 6.13: Fixture Design Representation with Face Tag Association	157
Figure 6.14: Typical Output of Design Change Detection of Affected Shape Entities	158

LIST OF TABLES

Table 4.1: Fixture Element Group Database	97
Table 5.1: Size reduction tests with model compression	125
Table 5.2: Timing tests for visualization	126

SUMMARY

Distributed Collaborative Design is an area within the collection of systems and methods for the digital modeling of the global product development and realization process. Global enterprises face tremendous challenges in collaborating together to design and develop products across geographically dispersed locations known as distributed environments. As design is seldom right the first time and requires design changes, global enterprises need to be able to synchronize information with one another. This thesis advocates that design synchronization involving design change is a key challenge to effective product-process interactions and early collaborative decision-making.

Successful distributed collaborative design involving design synchronization is not easily achieved with conventional design and manufacturing applications given that:

1. It is difficult to collaborate by exchanging entire product models in a seamless, integrated and flexible manner with these applications and yet avoid data proliferation and inconsistencies, especially when frequent design changes occur and timely, accurate and consistent updates with collaborating companies are needed.
2. The nature of geometric modelling or CAD systems is such that boundary representations provide important references to shape entities of the product model but it is not easy to share and manage these references consistently during design changes across collaborating users.
3. The lack of robust methods to detect design changes.

The research conducted in this thesis presents solutions to the above difficulties. A distributed collaborative design computing environment is needed as a key technical approach. Its foundation comprises a proposed middleware framework and an application architecture to support distributed product and process modeling in a seamless flexible manner. The middleware framework defines flexible services and mechanisms to product modeling capabilities and data for distributed environments. The application architecture is the definition and arrangement of architectural elements due to an appropriate distribution of functionality and data involved in distributed product design and development. This distribution conceptually helps to define the design synchronization mechanisms needed as middleware mechanisms to manage functionality and data issues to realize timely, accurate and consistent updates.

This approach allows suitable applications belonging to users and companies to be developed as application views integrated into the distributed collaborative design environment. Applications as such can access, develop and collaborate on product models based on a product modeler server providing necessary services related to product modeling and product data representations. An interactive fixture design application view illustrates this without having to have a full-fledged resident CAD system.

Given that design changes in product-process design interactions in distributed environments affect collaborative decision-making, it is crucial to deal with how they can be properly detected and updated to application views. Such capabilities are necessary to ensure product models and their changes are shared and referenced consistently and accurately. Otherwise collaborative decision making will be difficult

even as useful application relations may be logically set up to enable application views to relate to one another. Therefore design synchronization capabilities must manage application view integrity to support application relations. These capabilities have to deal with two major aspects of application views in distributed environments:

1. Providing updates to the 3D product model view and
2. Ensuring that product data representations referencing the product model are accurate during design change.

Accordingly, an evaluation of 3D graphics simplification techniques is needed. From the product design perspective, the integrated use of an enabling technology in 3D graphics compression is featured. This ensures that complex faceted data or models resulting from a product modeller can be compacted and updated to application views without compromising product model interpretation.

The most challenging aspect of design change deals with shape modifications which result in boundary representation changes that must be explicitly captured at the product modeller. Boundary representations comprise shape entities that are topologically and geometrically defined to enable robust product modelling. All vital information related to shape entity changes need to be captured and appropriately updated to application views. This is more effective compared to having to deal with the entire product model. Thus design change detection and update to application views driven from the product modeller is a vital aspect of design synchronization. This will ensure that all application views have the opportunity to carry out early collaboration consistently and accurately whilst avoiding unnecessary additional problem solving.

Chapter 1

INTRODUCTION

1.1 Background

It is rare nowadays for a single enterprise or company to design, manufacture and supply an entire product in a single location. The dominant situation today is that each enterprise or company concentrates on their core competencies and needs to dynamically collaborate with other companies in virtual value chains so as to meet today's challenges of product development. In this context, the abilities to design dynamically in a responsive manner, drive product development through the supply chains and manage costs are increasingly demanded. Consequently, this is technically challenging as appropriate support tools, resources and approaches are needed to address issues that impede the distribution, integration and support of various design and process activities to enable collaboration in distributed environments.

When a product is designed through the joint and collective efforts of many designers, the design process may be called collaborative design [Wang *et al* 02]. This may include functions as disparate as design, manufacturing, assembly, test, quality and even purchasing from suppliers and customers. Since a collaborative design team often works in parallel and across distributed heterogeneous environments in both asynchronous and synchronous modes, the resulting process may even be called a distributed collaborative design process.

Traditional approaches of sharing design information among collaborators and tools have included the development of integrated sets of tools and the establishment of data standards and formats for product exchange. These are, however, becoming insufficient to support collaborative design practices due to factors such as highly distributed design and process teams that need to be appropriately connected and synchronized; diverse heterogeneous engineering tools that continue to pose problems to integrated collaboration; fundamental shortcomings of present conventional systems even in supporting early design changes (as a design is seldom right the first time and customer satisfaction requires alternative innovations); and evaluation associated with frequent *ad-hoc* collaborations in an increasingly ‘outsourced’ and fragmented global environment. Such design changes are more likely to take place at the stages of conceptual design before detailed design. Overall, conventional CAD tools do not quite address the challenges of conceptual or early design especially in a distributed collaborative environment.

Technically, collaborative systems can be defined as distributed multiple user systems that are both concurrent and synchronized [Bidarra *et al* 01]. *Concurrency* involves management of different processes trying to simultaneously access and manipulate the same data. *Synchronization* involves timely updates through propagating evolving data among users of a distributed application, in order to keep their data consistent and improve responsiveness.

Concurrency and synchronization are generally demanding concepts. Their difficulty becomes particularly apparent within a distributed collaborative design context where large amounts of model data and design changes and flows have to be consistently

handled so that users can carry out their activities and collaboration in design and related processes.

1.2 Motivation & Purpose

Distributed environments are generally heterogeneous given different or diverse practices and systems environments. It is one major reason hampering the seamless integration of compatible product and process design capabilities, information and methods. A suitable approach is thus needed to conceptualize and develop an appropriate application architecture supported by a middleware framework to develop a suitable distributed collaborative design computing environment. To do so, there are also specific needs for various middleware to provide the relevant mechanisms and capabilities that appropriately distribute and synchronize functionality and data across the network [Bidarra *et al* 01] [Wang *et al* 02][Huang and Mak 03][Wu and Sarma 04].

This thesis attempts to address the following important issues

1. Development of a distributed collaborative design computing environment with the appropriate application architecture and middleware framework, and
2. Development of design synchronization mechanisms supporting timely, accurate and consistent updates to applications so that collaborative decision making can be based on handling design change.

In this approach, specific capabilities to demonstrate such a framework include the realization of interactive fixture design and product data representation as part of an application view in a distributed collaborative design environment. Capabilities for design synchronization across distributed environments are required to accurately and

consistently enable distributed product-process interactions and collaborative decision-making. It is crucial that these capabilities must be reliably driven by the product model's boundary representation so that design synchronization with updates is directly and generally possible. In reality, the nature of product model definition, as in its boundary representation and the accessibility and persistency of shapes and features within, has to be apprehended for successful collaboration between applications in product-process, and indeed product-product, interactions. The research approach has to overcome the weaknesses or shortcomings of conventional systems by exploiting Open Source technologies in order to facilitate seamless meaningful integration as a middleware framework would require.

Related to this thesis, distributed collaborative design is a key technical area within the research trends and perspectives grouped as Digital Enterprise Technology (DET) - 'the collection of systems and methods for the digital modeling of the global product development and realization process, in the context of lifecycle management' [Maropoulos 03].

It is also about conceptual or early design, crucial in the product development cycle in which the impact of early design decisions on manufacturing costs is initially very high, and declines steeply as the design matures. Thus the opportunity cost is very high at the preliminary design stage since subsequently, it is extremely difficult to compensate for a poor design. Conceptual design, guiding design definition and editing, in traditional Computer-Aided Design (CAD) is hard to accomplish, attested by the fact that most conventional CAD systems primarily focus on detailed design. The basic problem is that conceptual design requires knowledge of design

requirements and constraints which are usually imprecise and unavailable early on. There is considerable research on semantic feature modeling to enable better design intent and shape generation capabilities in CAD systems. Notwithstanding this, the key impact of conceptual design is surely in arriving at better product designs or design alternatives, through changes to the product shape definition and specifications, which would subsequently also affect detailed design processes. So in essence, design change involving shape editing is a key driver activity. In today's distributed environment context, conceptual design needs to adopt a more pragmatic and aggressive approach - *through collaboration* - supported by information technologies and the appropriate integration methodology with design [Wang *et al* 02].

Ultimately, as more advanced utilization is sought of the Internet as in twinning respective application and infrastructure areas respectively such as distributed collaborative design and Grid computing, middleware capabilities increasingly become more specialized and are driven by domain and context requirements at the top of the 'network stack'. Once such specialized capability is that of design streaming wherein likely, design changes can be streamed and shared incrementally without dependence on *a priori* complete or filed product models. A plausible approach to design streaming is to capture Boundary Representation-related topological operations underlying the design change for transmission and reconstruction in order to share and collaborate. Handling design change for synchronization that can also include design streaming is thus an important area of research pursuit. These specific capabilities require domain-specific approaches to enable and support scientists and engineers to transparently use and share distributed resources such as computers, data, networks, and remote instruments (or equipment) [Blatecky *et al* 02].

1.3 Organization of the Thesis

Chapter 2 reviews the background and industry developments in manufacturing and the resulting need for Digital Enterprise Technology cornerstones to meet the emerging challenges of product development and manufacturing known as Distributed Collaborative Design. It also reviews the relevant literature of related work reporting on propositions and developments of methods, techniques, applications and systems to satisfy the need to provide for collaboration and synchronization. Design change is highlighted as an important issue. Based on this, a problem definition is identified accompanied by specific objectives of this thesis.

In Chapter 3, a critique of conventional CAD systems is elaborated to highlight salient issues involving persistency of reference tags to geometry elements in the product model. Coupled with a set of insights, the framework design and its application architecture's elements are proposed so that middleware and integration issues can be resolved to enable domain users such as designers and engineers to collaborate independent of or decoupled from proprietary architectural and systems interfacing and integration issues. Resulting from this, application development and interactions can be supported such as in interactive fixture design. This approach can also allow further application development such as extensions to be possible. Chapter 3 presents the application architecture and systems environment for Distributed Collaborative Design.

Chapter 4 presents an implementation of the system environment with a demonstration of interactive fixture design capability based on it. A relevant comment is that

interactive fixture design may be treated as an assembly design activity guided by knowledge such as rules without necessitating design change involving shape editing.

Chapter 5 evolves the framework further with integrated model compression as an enabling technology to provide a key middleware mechanism for application view updating. This is supported by the importance of being able to drive local compression of faces from the boundary representation due to design changes. This leads to the need to investigate design change detection to update of all affected shape entities in a boundary representation. This is important to the maintaining the integrity of product data representations in application views. Several examples are included to demonstrate these design synchronization mechanisms to improve the middleware framework and make the application architecture appropriate to design change.

Chapter 6 provides further illustrations of design change detection and updates to application views. A case study involving fixture design and re-design due to design change is used to collectively cover the developments in this thesis. A critique of application relations management is made with design change detection and update to describe why design synchronization has to be design change-driven from the product modeler, such that application relations management and early collaborative decision-making would be more effective. In particular, the importance of persistency and consistency of referencing the product model's boundary representation during design change is emphasized.

Chapter 7 concludes this thesis with the main contributions made and the recommendations for future research.

Chapter 2

LITERATURE REVIEW

This chapter discusses the relevant literature. Section 2.1 reviews the historical background of design and manufacturing applications to highlight today's challenges. Section 2.2 provides theoretical cornerstones of an emerging perspective of Digital Enterprise Technology (DET). Section 2.3 describes DET functionality issues. Section 2.4 reviews related work relevant to distributed collaborative design identifying challenges and drawbacks of current approaches. Section 2.5 provides the problem statement and objectives. Based on efforts led by the author to develop a distributed computing environment with an applications architecture and a middleware framework, the focus of design synchronization is highlighted with regards to the context of design change affecting collaborative decision-making.

2.1 CIM, CE, CAPP and GLOBAL MANUFACTURING

Traditional research efforts in computer-based methods for design and manufacture largely relate to applications in CAD, whilst research in Computer-Automated Process Planning (CAPP) has substantially enriched design knowledge with concepts from the manufacturing domain. It is well recognized that applications developed in isolation would not promote the concept of Computer Integrated Manufacturing (CIM). The goal of CIM was to achieve the local network integration of systems [Maropoulos 99]. However, CIM did not enter the mainstream due to its high level of

complexity, costly infrastructure and poor support. Crucially during this period, competing standards in communication protocols also prevented flexible integration.

The complexities of product development and manufacturing practices also brought on the concept of Concurrent Engineering (CE). CE systems aimed to reduce ‘time to market’ through a simultaneous approach to product and process design [Sohlenius92]. This is facilitated through using Design for X (DFX—where X stands for any product life cycle phase) [Ulrich 00]. A key CE requirement has been for CAPP to enable production method selection, based on process capability and production economics, through automatic interpretation of design data.

However, a key issue is that highly detailed designs are needed before CAPP systems can perform their ‘micro planning’. This makes CAPP rather unsuitable in today’s context of rapidly evolving product designs and agile deployment of manufacturing resources. The global trend of reduced product lifecycles, increased product variety and cost competition has also placed strain on the integration of design with distributed manufacturing operations. Indeed, the concept of CAD itself has evolved to be tightly integrated with Computer-Aided Manufacturing (CAM) and Computer-Aided Engineering (CAE).

Notably the emergence of the Internet is due to the standardization and open adoption of primary data communication protocols. This is essential to supporting various specialized middleware capabilities and services. Standardized data protocols have also spawned new industrial segments in computers and networking. Notwithstanding this, the sense of isolated applications unable to work together is a

challenge to enterprises with dispersed activities in engineering design, fabrication, production and final assembly. Such enterprises are said to require or have structures or frameworks with the notions of design anywhere, fabricate anywhere, and produce and deliver anywhere. It is an expressed global vision to optimize available resources and deliver quality products, in a timely manner while maximizing profits [Reiter 03].

To support such a vision, a renewed understanding is that globalization forces have dispersed economic activities and outsourcing where components and intermediate goods are even shuttled between plants and countries for comparative advantage. With intense global competition, such activities can be conceptualized as transient horizontally fragmented value chains of interacting product design, planning and management and realization phases.

To trace this, Japanese car manufacturers had to achieve cost and quality competitiveness through highly efficient in-house production processes in the 1980s. The efficiency was due to factors such as vehicle platform sharing, parts modularity and interchangeability, and stringent quality controls in tolerance for parts assembly to contribute toward customer satisfaction. Subsequent to this, the leading manufacturer, Toyota strategically initiated its own parts supply chains of subsidiary companies, “*keireitsu*”, to lower costs of production resources and activities. This took place across lower tiers of supplier companies and later encouraged greater design and development autonomy for continual improvement. Such chains were however vertically integrated and dedicated to Toyota for it to concentrate on core product innovation, design and development in the key areas of engine efficiency, noise, vibration and harshness contributed by vehicle chassis design, build and overall

assembly. Other companies then quickly learnt from this strategy. However, customer demands continued to drive greater product sophistication and complexity adding pressure to manufacturing costs. This affected companies down the dedicated “*kereitsu*” chains. They could not easily remain cost-competitive and had to compete in other markets and overseas. In general, contract manufacturing came into being; the “*kereitsu*” chains did not last. Contract manufacturing nowadays is characterized by sizeable engineering teams, production capacities and with even more competitive downstream supply chains. This is to capture businesses worldwide from brand name owners or customers on higher tiers of the value chain focusing as well on a product’s assembly and critical parts, rather than just more ordinary parts.

Horizontally fragmented value chains would become the next phase as products now have very short life-cycles and even greater complexity. Brand name owners have become prepared to require contract manufacturers to become original design manufacturers with product development and final assembly capabilities. This frees brand name owners to compete with agile product design innovation and marketing strategies and efforts reinforced by intellectual property protection in order to survive in global markets. With all of this, highly fragmented, dynamic and fiercely competitive horizontal value chains now prevail.

The above elaborated trends highlight a new competitive environment in which product design and development activities are highly demand chain-driven priorities, *i.e.* customer-based, and time- and cost-sensitive, versus just supply-chain oriented, *i.e.* parts and components supplier sourcing and logistics efficiencies. Two key characteristics of these activities are:

- a) Activities are to stay ‘connected’ across geographically dispersed locations, technically called ‘distributed environments’.

- b) Activities are now even more change-driven, especially in what is called ‘design change’. A design is seldom initially right and with product variety and innovation, frequent design change occurs which requires capabilities in design synchronization and collaborative decision-making.

Synchronization is more than just the storing, retrieval and sharing of design data; it is the coordinated requirement of having timely updates propagated ‘across the systems’ to handle system and application inter-dependencies.

Finally, concepts such as Distributed Collaborative Design and Integrated Product-Process Development (IPPD) are increasingly vital as synchronizing distributed product-process models with frequent design change become a challenge. This effectively calls for seamless integration methods and mechanisms to distribute and support ‘connected’ applications. With complex relationships and requirements between customers and suppliers in the value chain, the lack of such approaches and architectures would always incur considerable costs. There is thus a call for new approaches and architectures in the underlying modeling and information management systems to support conceptual design, and manage early design changes across distributed enterprises [Lutters *et al* 01].

2.2 Digital Enterprise Technology (DET) Cornerstones

DET is defined as ‘the collection of systems and methods for the digital modeling of

the global product development and realization process [Maropoulos 03]. DET perspectives and research priorities call for the fundamental development of methods and systems focused on five theoretical cornerstones or technical areas (Figure 2.1):

1. Distributed and collaborative design
2. Process modeling and process planning
3. Advanced factory equipment and layout design and modeling
4. Physical-to-digital environment integrators
5. Enterprises integration technologies

DET requires synthesis of these five technical areas, of which the first three are in the digital domain and the next two in physical deployment. They interact with one another requiring feedback to distributed product development and realization teams.

Product design within a collaborative and distributed network is the first technical digital domain cornerstone utilizing the enhanced graphics and computer processing technologies as well as the communication infrastructure of the Internet. Of this, relevant (sub) issues include Distributed co-design, Design knowledge management and representation, Integration of design with manufacturing planning and Product lifecycle management.

Arguably, these issues are also related to the increasingly important role of conceptual design in product development even though design requirements and constraints are still usually imprecise [Wang *et al* 02]. At this early phase, conceptual design issues are also highly inter-disciplinary and involve collaboration from customers, designers and engineers in practice. These issues have significant impact on manufacturing

productivity and quality affecting downstream processes and tools such as machining, fixture planning, mould design and casting (Figure 2.2).

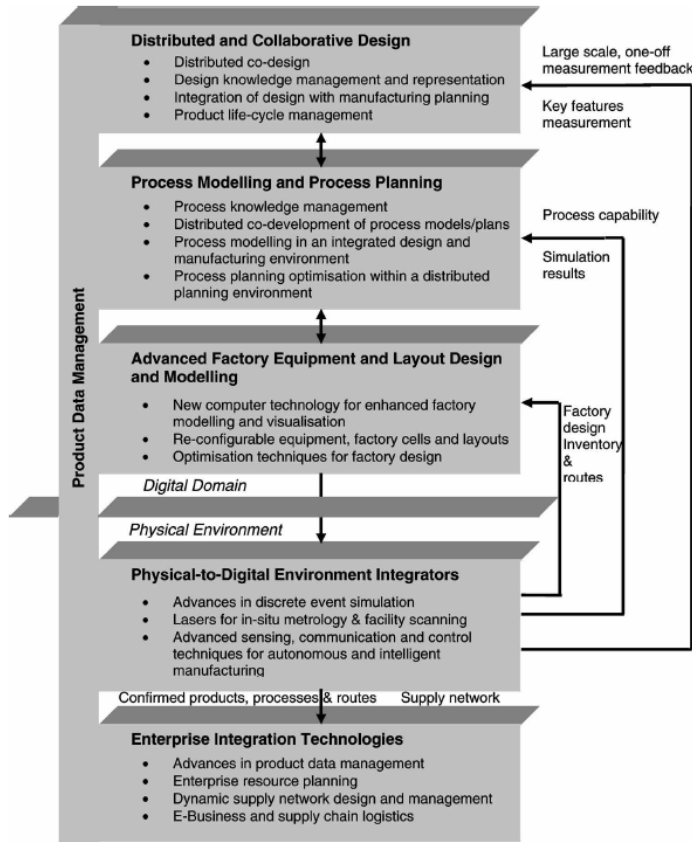


Figure 2.1: DET Theoretical Cornerstones [Maropoulos03]

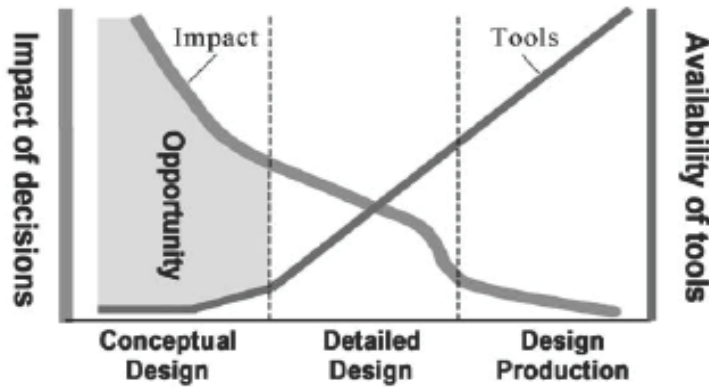


Figure 2.2: Importance of Early Conceptual Design Decisions [Wang *et al* 02]

Furthermore, [Wang *et al* 02] conducted an extensive survey of state-of-the-art research, projects and applications in the collaborative conceptual design domain, based on Internet and Web technologies, to identify future research trends. Commonly noted has been the realization of early design opportunity and its associated opportunity cost in terms of its manufacturing costs, notwithstanding the emergent distributed collaborative design research context.

Wang also observed that there exist many commercial CAD systems that support detailed design and *if at all*, few commercial tools support conceptual design at the boundary with detailed design (Figure 2.3). This can also reflect the paucity of general feature modeling and semantics in such conventional systems. They and/or their underlying technologies are not completely available today especially in the early stages of design and collaboration in distributed environments [Wang *et al* 02b].

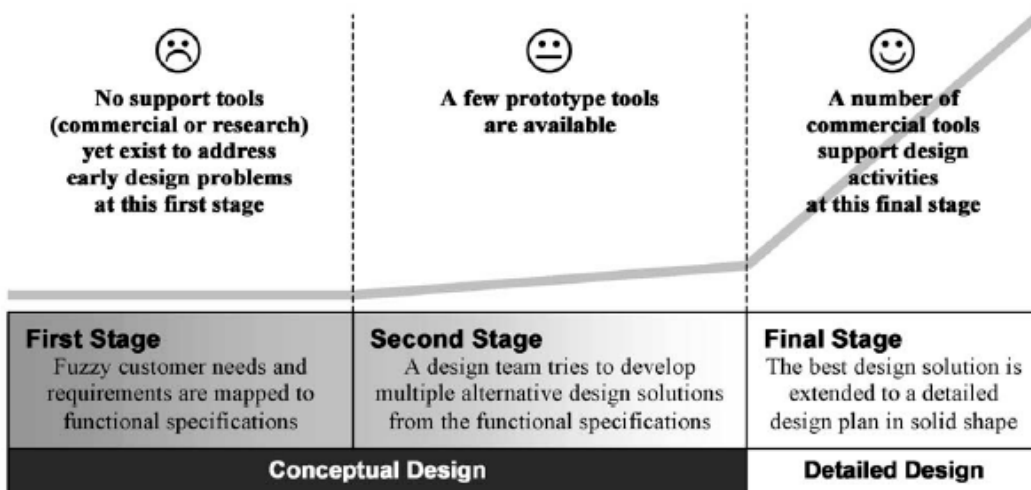


Figure 2.3: Availability of Design Tools [Wang *et al* 02]

Similarly, [Huang and Mak 03] evaluated topics and works related to product design and manufacturing given the importance of the Internet and WWW technologies to

manufacturing. They highlighted evolving interests from electronic commerce and business toward product development and shop floor processes, as the beginning of the digital manufacturing enterprise era. Many gaps are however found in the development and application processes due to domain and technological complexities. A simple example is the difference in graphical user interfaces between Web and traditional applications.

[Huang and Mak 03] also highlighted challenges to the operation, development and deployment of web applications. In particular, good consideration is needed to break down frequent user-system interactions into 2 phases: between the user and the client side system; and between the client and remote server. When interactions between server and client machines are kept at minimal levels with careful allocation of computation among them, high interactivity can be achieved through client side processing. This is a key consideration in the distribution of data and functionality amongst applications deployed as clients and servers.

[Li and Qiu 06] surveyed state of the art technologies and methodologies in collaborative product development systems classifying the levels of interactions and system infrastructures and complexity of enabling information technologies. Classifications ranged from purely visualization-based collaboration to facilitate product preview/review, to collaborative design capabilities in concurrent engineering-based collaboration requiring integration with manufacturability evaluation and simulation capabilities for lifecycle consideration. In future trends, they identified a major need to overcome system weakness in interactivity for real time effective collaboration. This requires effective distribution/collaboration

techniques through new methodologies to improve communication and cooperation.

DET deployment is characterized by a flat, 'heterarchical structure', with functionality configured by flexible integration of data repositories, distributed systems and user sites. The unique Internet infrastructure is also 'heterarchical' as the effective backbone for DET deployment, with key data communication and exchange standards such as STEP and XML (eXtended Markup Language). It is noted that XML is far more pervasive, expressive and open than STEP with its own limitations. Notably in the area of process modeling, the NIST Process Specification Language (PSL) Project proposes to standardize an XML framework [Schlenoff 00].

2.3 Digital Enterprise Technology (DET) Functionality Issues

Although the Internet provides the medium for data transmission and exchange, there are significant challenges facing the digital enterprise [Reiter 03]. The relevant ones include: Applications Compatibility; Data Management; and New Releases and Proliferation of Software Technology and Implementation.

These challenges recur with each new technology implementation. An example is the implementation of solid modeling. Initially (late 1980s) this technology was very expensive and considered a risk that was hard to use and justify. As solid modeling technology matured and received widespread acceptance, its negative aspects initially impeding its proliferation largely vanished. Justification for hardware and software for each solid modeling seat then also became a minor issue.

On the challenge of compatibility, it is in the nature of manufacturing and software

industry practice that systems integrators and application software providers have to rely on external programming interfaces to produce dedicated but costly solutions between pairs of systems. Further it is also time consuming to carry out effective exchange of information between new partners.

The manufacturing software industry thus became traditionally characterized by dedicated, integrated product design and manufacturing applications. These heavily integrated systems, more recently branded as Product Lifecycle Management (PLM) solutions, provide a suite of design and manufacturing applications and the necessary mechanisms for information exchange. However, the applications are mainly standalone applications from legacy. Examples include those by UGS [UGS PLM Solutions, 2004] and PTC [PTC PLM Solutions, 2004]. Based on these systems, there is no need to employ the services of systems integrators to develop customized mechanisms. However, the drawback is that companies are often required to use applications from the same PLM vendor before they can exchange information. This becomes a problem in a heterogeneous environment when companies collaborate with new partners who do not use applications from the same vendor. Further, it is unlikely a PLM vendor will supply all the different product and process design applications needed by different enterprises. In addition, sometimes even the same applications from a vendor may not integrate well the models from these applications to ensure consistency. For example, PTC provides Pro/CONCEPT to carry out conceptual design, in addition to Pro/ENGINEER, but maintain the consistency between the model for conceptual design phase and the models for other design phases.

Similarly today, distributed collaboration, cooperative and distributed design, and the

related synchronization of Internet-centric design and planning systems are highlighted as new research challenges [Maropoulos03]. The lack of DET functionality for the early rapid evaluation of planning options is a key constraint, severely limiting synchronization with design and support for sourcing decisions during early product development. An intrinsic problem is the over-reliance on traditional feature-based CAPP/CAM methods that are more effective during re-design and detailed design. Vice-versa, the paucity of information during early design may not allow feature-based planning methods to function in a reliable manner, a point reflected by [Wang *et al* 02].

DET deployment goal is the scalable and re-configurable integration of distributed functions/data, and coordination of design/development teams in any enterprise.

2.4 Related Work

Manufacturing application development is carried out mainly in two ways. One is based on a standalone CAD system's application programming Interface (API) exposed to users. A multitude of dedicated and proprietary functionality is included in such systems and familiarity is required with each system's design. The availability of a CAD system API is more motivated by users' specific needs to exploit the system. Notably, agents were used but the interaction could only be based on sharing and communicating codified knowledge across disciplines in order to integrate systems [Cutkosky *et al*, 93].

For better integration and other key reasons, another approach is to basically build applications directly with solid or geometric modeling kernels. A notable approach

was reported in [Han and Requicha, 98] in using a kernel as a geometric modelling server. Most conventional CAD systems have had this approach and have become known or evolved as standalone monolithic and complex modeling systems [Hoffman *et al* 98].

Based on such standalone systems, basic approaches to an integrated environment for product and process design can include rudimentary use of standard file formats such as STEP and IGES for CAD models located at central databases. [Roy *et al* 99] proposed a World-Wide Web (WWW)-based collaborative design framework but it requires a translator to convert CAD models into neutral VRML models stored in a remote product data repository for remote viewing. The translator resides on a central server to be accessed remotely by a designer.

A number of information-oriented frameworks [Pahng *et al* 98] [Huang *et al* 99] have also been proposed and are regarded as under proof-of-concept development stage purposed on an application [Wang *et al* 02]. [Xie *et al* 01] proposed a WWW-based integrated sheet metal product development platform based on an information integration framework to link part design with process planning, simulation and manufacturing systems. But the part geometry has to be represented in STEP files.

Additionally, [Huang and Mak 03] investigated how a web application itself can be developed for managing engineering changes. Accordingly, engineering changes are a kind of modification in forms, fits, functions, materials, dimensions etc of products and constituent components. Indeed, the agility of an enterprise today depends on its ability to manage changes efficiently and effectively. Engineering change

management therefore has a direct impact on the enterprise's product development process. However, engineering changes involve tremendous complexity affecting systems such as in CAD, CAPP, Product Data Management (PDM) and Enterprise Resource Planning (ERP). Although sophisticated computer aided systems with comprehensive functionality are available, such systems have not been utilized to facilitate engineering change management activities. [Huang and Mak 03] also pointed out that standalone computer aided systems are limited in supporting the multi-disciplinary teamwork in engineering change management, especially when they are geographically dispersed.

[Huang and Mak 03] thus proposed a web-based engineering change management framework to facilitate information sharing via web forms among various parties at disparate locations and also to achieve simultaneous data access and processing. It has basic functions such as request, evaluation, notification and logging of engineering change, to support management over distributed environments though relevant enterprise information is not incorporated and nor product design configuration or structure is not dealt with. It is part of the development of an engineering change management platform. They reported that the system scope can be extended to incorporate the facilities of conventional product data management systems that provide vault-like design file check in and check out capabilities. It has also been indicated that interfaces with systems such as CAD, CAPP, *etc*, need to be addressed.

In addition, [Huang and Mak 03] reported work on collaborative concept design to aid product definition, before design review and release management. It is a design tool to support collaboration on functional requirement analysis, concept generation and

concept evaluation. Morphological generation charts are used to choose combinations of concepts with evaluation based on selected criteria, quality function deployment and morphological analysis. The outcome is a preliminary layout design reflecting the product's working principles and features. Interfaces with CAD, CAPP etc systems are presumably also required to support design review and release management.

When collaborative functionality is designed as a plug-in or tied to separate standalone systems such as in CollIDE [Nam *et al* 98], ARCADE [Stork *et al* 97] and CSM [Chan *et al* 99] and the above, the resulting architecture requires users to have local private use and workspaces, and necessarily invokes onerous tasks of *copying model data* as files from local into common shared workspace for synchronization [Bidarra *et al* 01].

In such architectures, model data files would proliferate restricting the scope for collaborative design as design changes would occur when designers and engineers interact. A root cause is the problem of association and persistency of names (tags) to reference geometric entities. These references are internally generated by the CAD system or a geometric modeling kernel during runtime and are not automatically kept persistent and consistent. As such this problem is not resolved by translating standard file formats or copying model data files about. Each translation or copy effectively results in new model with different tags during runtime. Such issues also raise questions about the suitability of CAPP systems as process planning itself cannot easily evolve with design change. Another drawback in such architectures is that conventional standalone CAD systems used are already complex and monolithic, requiring much computational power [Bidarra *et al* 00].

Model data file proliferation involves frequent transfers of large amounts of model data across distributed environments. Despite tremendous improvements in its bandwidths, the Internet is a shared infrastructure connecting computers with a growing spectrum of new uses and applications. Indiscriminately transferring complex models and assemblies could always take much an inordinate and unpredictable time, an issue described as latency.

In addition, there is more relevant and related work with at least one distinctive, *i.e.* attempts at distributed computing and architecture either conceptually or with implementation efforts of developing distributed applications incorporating a geometric modeling server. Several observations will be indicated in association with architectural considerations such as conventional systems and geometric modeling servers; product model and data representation; as well as the construction of application views.

Several researchers have proposed the use of a central geometric modeling server for developing these distributed applications. [Han *et al* 98] discussed an approach that provides transparent access to diverse solid modelers for applications in a distributed environment. Solid modelers were augmented with software wrappers to provide a uniform API. Their system encompasses a feature-based design system, a central geometric modeling server supporting an automatic feature recognizer and a client-based graphics renderer. The geometric modeling server stores the B-rep model of a designed part. When a design change occurs, the design system communicates the change to the feature recognition system. One drawback of this approach is its dependence on form feature recognition. There is no product data representation on

the client side to support application views and processes; updates to the graphics renderer are only wireframe information extracted from the B-rep model.

[Martino *et al* 98] proposed the integrated use of design-by-features and feature recognition capabilities suggesting the definition of a homogeneous multiple view feature-based representation of the part model. This is called an intermediate model shared among various applications. However, while there has been research on feature definition for different domains, not all applications carry out reasoning based only on feature representation using geometric forms. The difficulty of feature mapping or conversion is thus highly context dependent and delicate in the wider context of distributive collaborative design involving early frequent design changes and limited detailed design information.

[Shyamsundar *et al* 01, 02] proposed a client-server architecture for collaborative virtual prototyping of product assemblies over the Internet. A polygon-based representation of the part was used for visualization and a compact assembly representation was also developed. A solid modeling kernel was employed as an application server to remove the complexity of installation and maintenance of the solid modeler on clients. Design changes are not automatically transmitted to users working on the model. However, assembly features are tagged and if a designer attempts to modify that face, the designer receives a warning.

[Hoffman *et al* 98, 00] proposed another approach for a product master model to unite CAD systems with downstream application processes for different views in the design process. They presented an approach to handling design change to synchronize

applications through the creation of associations across clients. Proposed is the Master Model concept with mechanisms for maintaining the integrity and consistency of the deposited information structures of these associations. It has several clients, with their own CAD systems, one of which is for a designer to changing the net shape (Figure 2.4). Net shape is one of the information structures in the master model. The other clients are domain-specific applications on CAD systems, dealing for instance with manufacturing process planning, geometric dimensioning and tolerancing (GD&T), cost estimation, performance evaluation, etc.

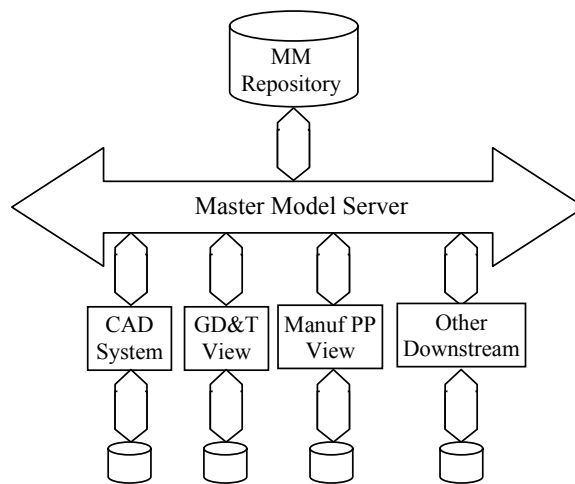


Figure 2.4: Master model architecture with client views

Hoffman noted that current approaches handle the consistency and association problem by organizing conventional systems as a limited *one-way* architecture. The features in an application view are derived from the features of the privileged view, usually the design view. The designer defines this view and conversion modules generate application-dependent feature models. If a modification is required by a downstream application, a privileged view must be entered after which new application dependent views can be derived from conversion. It is left open as to how to respond to design changes especially amongst heterogeneous CAD systems,

implying downstream information must be explicitly re-associated. To handle and coordinate consistent views has been noted to be a synchronization challenge.

It is noted that Hoffman did not provide the means and details to developing such systems [Bronsvoort *et al* 04]. Hoffman focused on existing CAD systems as the client and means to support process-centric views. This is despite the fact that the Master Model repository concept can be easily treated as a *separate* concern from a central geometric modeling server, perhaps due to the preservation of privacy and proprietary information associated with standalone CAD systems. Feature conversion inside CAD systems as design changes was not detailed, so too the means to detect and transfer these changes to synchronize with distributed application views. Hence, middleware concerns such as an appropriate middleware framework and synchronization mechanisms in a heterogeneous distributed design environment were not addressed. In using conventional standalone CAD systems, application development will have to depend on each system's proprietary API.

[Bidarra *et al* 01] developed WebSPIFF, a web-based collaborative feature modeling system. It is a client-server architecture where the server coordinates collaborative session, maintains a shared model based on a multiple-view feature modeling kernel [de Kraker *et al* 97]. All views are synchronized by feature conversion through the ACIS kernel-specific cellular model mechanism (CM) to support feature semantics [Spatial 02] [Bidarra *et al* 00]. The CM models the net shape by a refined cell complex. This permits editing from *any* feature view and eases consistency across all such views. But this requires complex shapes to be decomposed into numerous cells and is applicable only to the ACIS kernel. Feature conversion may be computationally

expensive requiring better server architectures [Hoffman *et al* 98] [Bidarra *et al* 02].

Related to the above, a framework for integrated part and assembly modeling has been reported [Bidarra *et al* 02]. But it incorporates a cumbersome ‘selection model’ - 3D objects representing canonical shapes of features in a given view. It is to support interactive selection of feature faces on a geometry image associated with modeling and an accompanying neutral VRML visualization. The drawback is that VRML is not editable resulting in an application is not truly interactive and integrated. From a product model and data representation perspective, VRML as a neutral representation gives the notion that it is a representation of the geometric model. Strictly speaking, this is not true as VRML was proposed as a static file format dedicated to 3D scene visualization or ‘publishing’ rather than ‘editing’ [Wang *et al* 02]. It has no provisions for dynamic integration to applications to support interactive collaboration with augmented data representations. No associative relationships amongst application views were reported to support collaboration. Despite the use of 3D faceted data sets in VRML, no synchronization support in terms of updates for view accuracy and consistency during design change was reported.

In the context of faceted/meshed models, [Wu and Sarma 04] reported on the importance of dealing with them in integrated design environments. They indicated that product models go through extensive shape modifications in various CAX applications before manufacturing. They identified the problem of repeatedly rebuilding the entire faceted model due to design changes to a solid model as involving costly computation to re-evaluate the B-rep model and re-generate the facets. It is even more so in distributed environments where typical transmission of

entire faceted models or product models could take up much bandwidth. They have not proposed a method to handle the problem of faceted model transmission given in their view that compression and progressive transmission techniques are amendable only to entire faceted datasets or models. Rather, they proposed an approach known as incremental facet editing for data exchange involving editing and evaluating the underlying B-rep model is adopted. This approach however requires dealing with 3 pairs of models for monitoring changes (that is pre and post states) to the underlying B-rep and faceted models on the application server-side and application client-side. These models have the cumbersome role of ensuring that only ‘changed’ facet models are transmitted for updating at other applications.

In comparison with the related works above, the basic aim of this thesis has been to propose and develop the appropriate middleware framework and application architecture to support distributed collaborative design in heterogeneous environments. A core understanding has been the distribution of functionality and treatment of data which may address the drawbacks highlighted above, such as proliferation of product models due to static file formats, reliance on cumbersome standalone CAD systems, reliance on feature recognition and conversion, insufficient approaches to address heterogeneous environments, insufficient or inflexible product data representations for application views to interact with product modelers.

Given the presence of B-rep solid modelers and mesh generation providing faceted models, this approach basically comprises (augmented) product data representations with XML with integration to a geometric modeling server to enable access and modeling from application clients. Faceted models are leveraged to provide for

interactive visualization and flexible manipulation of product models with their topological information. They are the means to maintain the product model similar to the master model server approach except clients do not need to have a standalone CAD system and the corresponding B-rep model. Unlike features, it is proposed here that the geometric elements of the design model in themselves are really primary to the needs of synchronization and design change handling for process domain-specific requirements. For example, a fixture design application often carries out reasoning based on the geometry of a part. Fixture elements are often associated with the faces of a part to access and carry out fixturing [Senthil kumar *et al* 01]. Through integration, a finalized fixture assembly can be formalized as the CAD model. This is a good approach given its independence of the geometric modeling kernel and static CAD file formats, and the ability to augment via XML.

Neither [Hoffman *et al* 00] nor [Bidarra *et al* 02] focus on augmented product data representations to support local interactive application view updates due to shape modifications. Hoffman's proposal is mainly an architectural proposal that relies on existing CAD systems to arrive at distributed product model management and Bidarra relies on semantic features tightly bound to a feature modeler server with specific cellular model topology support from the ACIS kernel. Moreover, the augmented product data representations are extensible to cope with product architectures or assemblies – a fixture design may be considered as an assembly configuration.

As such, the middleware framework and application architecture developed earlier have been extended to include design change management based on application relations between an application view's functionality as in fixture design, and the

product model data as in the faces for fixture element selection [Mervyn *et al* 03b]. Specifically, when design change occurs, it is relevant to the fixture design application to be notified. Application relations provide the means to monitor this and then activate independent problem solving on an application view for collaborative decision making. Generally, design changes can be said to be propagated supporting design synchronization at the application problem solving level.

Although [Mervyn *et al* 03] uses design change propagation propagated via application relationships with face shape entities, the underlying middleware does not have synchronization mechanisms appropriate to design change handling within the product modeler server to support shape modifications and application view updates. Notably, the product workpiece has to be re-imported into the product modeler server. Consider when dealing with the relationship of a face in a product model to fixture design, as in for example, locating face, during distributed collaborative design. When there is a design change, as in introducing a step or slot, there would be a need to automatically detect the geometric entities that have been generated, modified/replaced, removed or even re-mapped, and update the application views. This requires evaluating the B-rep rather than resorting to a re-import.

Further when addressing faceted models used for application views, and that are subsequently repeatedly rebuilt due to design change, simplification or compression techniques need to be considered. Compared to [Wu and Sarma 04], the model compression technique is leveraged into the middleware framework for design synchronization for timely accurate application view updates in relation to the augmented product data representation. Basically [Wu and Sarma 04] does not take

advantage of or rely on interactive design commands or shape editing procedures available to the modeling kernel and which are the means of creating and modifying product designs. Nor does it realize that ‘captured’ facet change models due to design change are amendable to model compression and could be used to update application views. In this context, the augmented Product data representation is used to incorporate the role of compression in design synchronization supporting collaborative decision-making across distributed environments.

In summary, the design change challenge requires synchronization mechanisms to enable a product modeler server to deal with shape modifications as deltas within an emitted B-rep representation and provide the corresponding faceted model updates in augmented product data representations for an application view to proceed with its albeit private problem solving methods.

2.5 Problem Statement and Objectives

A major research goal has been the development of a middleware framework and application architecture to address distributed collaborative design with a view towards design synchronization. On top of defining and developing appropriate architecture elements relevant to geographically dispersed settings, such a framework requires middleware mechanisms to support design synchronization in view of the domain and technical complexity of distributed collaborative design.

To be able to achieve distributed collaborative design, a middleware framework as motivated earlier, is required considering the following major problems today:

1. Compatibility problem: In today’s context, various different companies

collaborate to realize a product. The use of different software products often results in practical compatibility problems resulting in poor inter-operability. Such compatibility problems had cost companies about US\$1 billion per annum in the automotive industry alone [NIST 99]. This is exacerbated in today's context of distributed collaborative design demanding seamless integration and continuous collaboration.

2. Distributed information model and exchange problem: Information exchange is a critical component of a computing environment for distributed collaborative design. In IPPD, downstream applications of the product development process require the right information to carry out their tasks, while upstream applications require feedback information. In distributed collaborative design, these activities may entirely be *ad-hoc* collaborations reflected by the need for early engagements in dynamic fragmented value chains. Although a large amount of research has traditionally been conducted on developing information models for different applications, present day commercial applications often do not provide the required information.
3. Efficiency problem: Various design changes take place during product-process interactions in order to reconcile many requirements of other domains. Each time a design change or 'churn' occurs, applications need to retrieve the updated incremental information. For instance, if it is traditionally about retrieving large data sets, such as conventional CAD files, then this would be problematic, time consuming and unproductive as will be seen later on in addressing the effectiveness of conventional CAD systems. Several categories of design changes may be considered: 1) Shape changes; 2) Changes of parameters, dimensions, and constraints; and 3) Changes in attributes. Of

these, shape changes are the most difficult ones to respond since the rest may be considered more informational by nature.

4. Complexity problem: The creation of a monolithic system from the ground up is not only difficult, but also leads to difficult software maintenance problems. This is a complexity that can only be overcome if capabilities are flexible and properly distributed and collaborative.
5. Synchronization problem: In total, the above leads to the design synchronization problem in distributed collaborative design. In the expression of dynamic concurrent product design and associated processes in distributed environments, it is necessary that all applications are accessing and using the correct and coordinated forms of updated data. Emerging distributed collaborative design systems could provide remote collaborative viewing of parts and assemblies on thin clients albeit with proprietary formats for instance, compared to conventional standalone systems. However, the provision of middleware synchronization mechanisms for timely updates and design change handling across distributed environments has not been adequately addressed, especially to drive application relations and support collaborative decision-making. Today's design and manufacturing applications however work in isolation and a proper middleware framework is desired with specific mechanisms for effective integration and propagation of design changes due to the product model itself.

Synchronization here involves timely updates to maintain the consistency of all application views and information relating to the master product model as a result of design changes. This comprises two main integral considerations to ultimately

expedite early collaborative decision-making across distributed environments especially when design change involves shape modification occurs. In view of the latency and bandwidth constraints, one consideration is to provide middleware mechanism(s) that appropriately provide for accurate timely updates to support interactivity and view consistency. The other is to provide the accompanying support for design change handling at the shape modification level to expedite collaborative decision-making and problem solving. This is with reference to the associative relationship management capability in an Integrated Product Process Development (IPPD) context [Mervyn *et al* 03b].

A middleware layer perspective has been adopted with the concept of understanding the distribution of data and functionality in product design and manufacturing to conceptualize and develop an application architecture and computing environment. Next, synchronization across distributed environments requires the timely update of primary and generally complex 3D design and dataset information to be efficiently transmitted or transferred and updated as design (change) itself happens remotely [Bok *et al* 04]. This should be non-destructive in order to ensure the integrity and accuracy of presenting the design and design changes for collaboration. This is also primary to the need to support collaborative decision-making and problem solving in response to design change.

For example, in a change-driven collaborative design environment, a design engineer needs to dynamically carry out design changes to a product part or component whilst possibly and concurrently requiring the engagement of a ‘downstream’ supplier of tooling, *e.g.* fixture design, to carry out some form of processing to derive a new

consistent application view of the product data. The processing is therefore application-bound, domain-specific and may employ different methods preferred by, peculiar to, or even proprietary to the practices of say, fixture design, in a company. In addition, one area of concern is to develop and integrate simulation capabilities in order to interact with fixture design and analysis for a more complete collaborative product development. In other words, such interactions would become further extensions of the framework supporting product-process and process-process collaboration. This then allows for a more concurrent engineering-based type of collaboration as highlighted by [Li and Qiu 06].

The conceptual development of the middleware framework and application architecture has been led by the author. This provides the groundwork for research into design synchronization issues affecting product modeling and product-process interactions. This required original conceptual and technical contributions by the author including:

1. Development of geometric modeling server through a solid modeling interface to demonstrate distributed interactive and portable access into geometry using Java Native Interface, Java Remote Method Interface, Java3D and Parasolid (as opposed to a conventional standalone CAD system).
2. Development of product model data representation to demonstrate the feasibility of application views through XML modeling of B-rep information (at the levels of face, edge, vertex and point for selection) and the associated 3D facet data for interrogation and manipulation (as opposed to relying on VRML).
3. Further development on the application view of interactive fixture design

methods through reusable client classes to demonstrate the feasibility of reusability and distributed design (to allow for modular and independent integration of different process methods to respond to design information).

4. Development of modeling classes to demonstrate interactive design capabilities affecting the B-rep model (to show that interactive remote shape generation is possible).
5. Demonstration of integrated model compression as an enabling technology for complete design or solid models for the handling of large complex datasets in association with product model data representation (to handle bandwidth limitations across distributed environments in general).

In summary, for design synchronization to take place integrally, firstly, a product model and its design and the subsequent changes should be appropriately supported by a distributed application architecture and its elements especially a geometry modeling kernel and an application view; secondly, mechanism(s) for transmitting and propagating updates to remote applications views should be provided to expedite collaboration; and thirdly, as a result, application-bound or dependent algorithms may deal with these updates for problem solving.

This thesis aims to solve these problems by developing a distributed collaborative design system with middleware mechanisms to support collaboration during product-process interactions involving design change. The thesis has the following objectives:

- Conceptualise and develop a middleware framework with appropriate basis of distributed functionality and data for architectural elements to support distributed collaborative design.

- Develop the computing environment involving these architectural elements with the appropriate functionality and data.
- Demonstrate how an application such as interactive fixture design across distributed environment can be supported based on the appropriate distribution and representation of product modelling data and information support and functionality amongst these architectural elements.
- Develop design synchronization mechanisms appropriate to distributed environments to enhance the framework to efficiently support timely accurate updates to application views and provide design consistency in relation to design change and early collaborative decision-making across application views. This can facilitate process application views in responding early to design changes during product-process interactions and so avoid unnecessary additional problem solving.

Chapter 3

MIDDLEWARE FRAMEWORK AND APPLICATION ARCHITECTURE FOR DISTRIBUTED COLLABORATIVE DESIGN

The issues and requirements defined earlier have necessitated the design of a middleware framework and application architecture for the development of a computing environment for distributed collaborative design. This chapter has the objective of addressing this need. As a backdrop, a critique of conventional CAD systems in relation to distributed product-process interactions in the distributed collaborative design and design synchronization context is provided. This is accompanied by insights into the importance of distributing functionality and data from the perspectives of integrating product-process interactions and supporting design synchronization in distributed environments. This importance reinforces the need for middleware mechanisms to support design synchronization in collaborative decision-making. The framework accompanied by its architectural elements and the corresponding distributed computing environment is then proposed. To add design synchronization to the framework, the distributed product-process modelling context is illustrated highlighting two essential mechanisms in the research efforts so far: the leveraging of the model compression technique to synchronize application views for more timely and consistent updates due to the nature of distributed application view(s); and handling distributed design changes in the context of shape

modifications. These mechanisms are fundamental to a product modelling server in order to carry out collaborative decision-making based on application relationships driving problem-solving to assess design change impact on an application view such as fixture (re-)design.

3.1 Conventional CAD Systems

Almost all current feature modeling systems are based on parametric, history-based modeling systems, requiring a boundary representation (B-rep) as the main geometric model. Traditionally, the boundary representation can be used for several applications, *e.g.* process planning and computer-aided manufacturing or to even derive finite element models. Such systems include most if not all the existing commercial systems.

The basic entity in a *feature model* is the feature, defined as *a representation of shape aspects of a product that can be mapped to a generic shape and functionally significant for some product life-cycle phases*. Features should additionally support well-defined meanings, or *semantics*, for a particular life-cycle activity. However, beyond the default reliance on generic shapes, two important aspects of this are not well covered by most commercial systems. First, feature semantics as a form of useful knowledge for product design and development are poorly defined, limiting the capability of capturing design intent in the model in conceptual design. Second, feature semantics are poorly maintained, permitting previous design intent to be inadvertently overruled. Such systems are said to lack *validity maintenance* facilities [Bidarra *et al* 00].

History-based modeling systems are procedural systems which, together with an constantly evaluated B-rep, keep track of information about each modeling operation performed, *e.g.* the type of feature created, its parameter values, and its model references for positioning. The stored sequence of modeling operations, called the *model history*, completely determines the B-rep. Each new feature is positioned relative to boundary entities of the evaluated model, obtained from previously created features. Creation of a feature produces in the evaluated boundary model its characteristic shape imprint.

Feature instances can be modified by specifying new values for their *parameters*, or be deleted from the model. This is done by modifying, or deleting, the respective feature creation operation in the model history, after which a newly evaluated boundary model is created by sequentially re-executing the operations in the modified history. With this scheme, variants of a feature model can easily be created, for example, variational constraints-based geometry.

However, most current feature modeling CAD systems have six major shortcomings [Bidarra *et al* 00]:

1. Computational Cost,
2. Non-Associative Set Operations,
3. Entity References,
4. Constraint Solving Limitations,
5. Persistency, and
6. Feature Semantics

The first three shortcomings are fundamentally due to the strict dependency on the historical order of feature creation. The fourth is due, in particular, to non-bidirectional dimensional constraints, causing the model to be rigid. The fifth is notably due to the historical evolution of entities in the evaluated boundary model adversely affecting the persistence and definition of feature semantics resulting in the poor tracking of changing references to topological entities. The last shortcoming is that of feature semantics support and maintenance, which is *non-existent* in conventional systems. The built in procedural validation schemes during conventional feature creation are limited and highly localized to a subset of the boundary model. Such non-global schemes easily cause previously created features to be invalid. Systematically analyzing the entire boundary model after each operation is also not feasible as there are little (or insufficient) traces of entities involved in preceding features. Current approaches prefer a more powerful declarative approach for modeling and validation of semantic features.

Intrinsically, the manifold B-rep does not allow storing of all feature information. This naturally excludes the possibility of analyzing the topology of the boundary of features, an essential requirement for detection of validity violations – called *feature interactions*. Feature interaction phenomena are regarded as a major problem affecting feature semantics [Regli *et al* 96], but are not at any rate dealt with in current systems. Thus, the design intent – the goal of feature semantics - expressed in user-defined feature semantics gets compromised. Therefore it can be said that current systems offer more of a geometric modeling approach comprising high-level primitives (*form* features) to create a boundary representation, rather than a genuine feature modeling approach.

Other than the abovementioned, supporting distributed collaborative design requiring design synchronization support involving design change or even exchange with conventional systems presents another related major shortcoming. To review, in a heterogeneous distributed environment comprising networked standalone systems, onerous static CAD file transfers and data translations/exchanges ('copying model data') would be typically required for sharing design information.

Traditionally, once a CAD system emits a standard B-rep of a design for translation into, for instance, IGES or STEP, the connection of the shape elements with the CAD model is lost – a major reason why data exchange such as STEP/PDES and file translation cannot adequately support the needs of distributed collaborative design involving continuous design changes. There are no links or associations 'across' provided for in such files to connect vital design details between users once such a transfer takes place, making it practically impossible to have persistency or consistency support.

The STEP/PDES standard may also falter, given that firstly, it focuses on describing and exchanging finished whole designs rather than on capturing and supporting early design changes (a condition similar to the role of CAPP highlighted in Chapter 2); and secondly, the nature of design and the corresponding issue of design feature semantics is rapidly changing due to an increasingly competitive industrial environment which increasingly imposes the need to be open, flexible and agile.

Matters are also exacerbated in today's context of rapid early design and changes since data transfers involving time consuming exchange and translation are onerous

and the resulting static CAD files have to be re-opened or re-instantiated in a design session as a standard evaluated B-rep model. This also means that only the conventional CAD model, stored in the native system's proprietary format, and not the standard B-rep model, can be edited conveniently [Hoffman *et al* 98]. However, such proprietary systems and their formats were also the reason behind the earlier development of exchange standards. Notwithstanding this, the problem of distributed collaborative design is compounded as typically, there is no built in support in such systems for persistency in names or tags of geometric entities. Perhaps this is also one key reason for many applications to be tightly integrated to a geometric kernel modeler. However, the resulting systems are still standalone.

In summary, where conventional CAD systems are concerned, distributed collaborative design involving comprehensive design synchronization support for managing design change is difficult to achieve. Even though advanced facilities are now available in high-end commercial CAD systems due to faster and more powerful hardware, this also results in larger and more complex standalone systems which impose a *premier* design view and a resulting *one-way* architecture [de Kraker *et al* 97; Hoffmann *et al* 98].

The fundamental problem of managing consistent core topological data identity involving naming and addressing tags to dynamically support and reference design entities is furthermore difficult to resolve. Next, with the reality of heterogeneous distributed industrial environments that have also resulted from outsourcing and fragmentation of value chains (Figure 3.1), requiring flexible collaboration and synchronization, it becomes necessary to consider a distributed product-process model

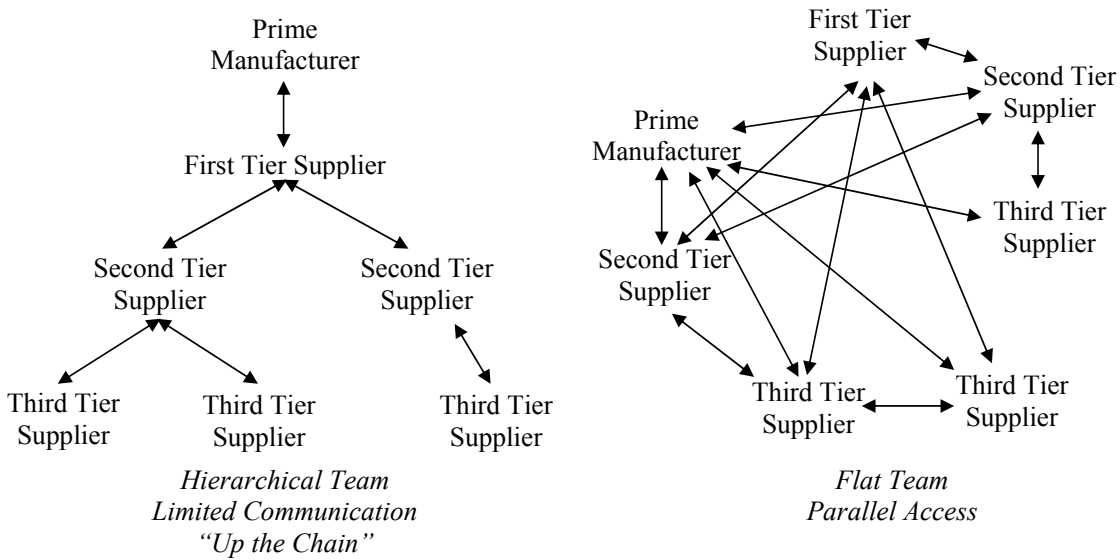


Figure 3.1: Distributed Industrial Environments - Vertical to Horizontal Fragmented Value Chains.

perspective and appropriate distribution of functionality and data to remotely handle design and design change as a CAD-CAPP-CAM perspective is also inappropriate nowadays. A distributed application architectural based on a middleware framework perspective thus needs to be conceptualized and designed with underlying seamless integration of product-process development capabilities is necessary.

3.2 Middleware Framework and Architectural Elements

A common need in a distributed collaborative design framework is to resolve the recurrent conflict of and the need for an appropriate distribution of functionality and data [Bidarra *et al* 01][Huang and Mak 03][Li and Qiu 06]. It would thus be helpful to draw out some domain insights and principles from classifying these functionality and data to help improve understanding of the architectural framework and the important context of addressing middleware concerns relevant to achieving distributed collaborative design capabilities. Figure 3.2 is referred.

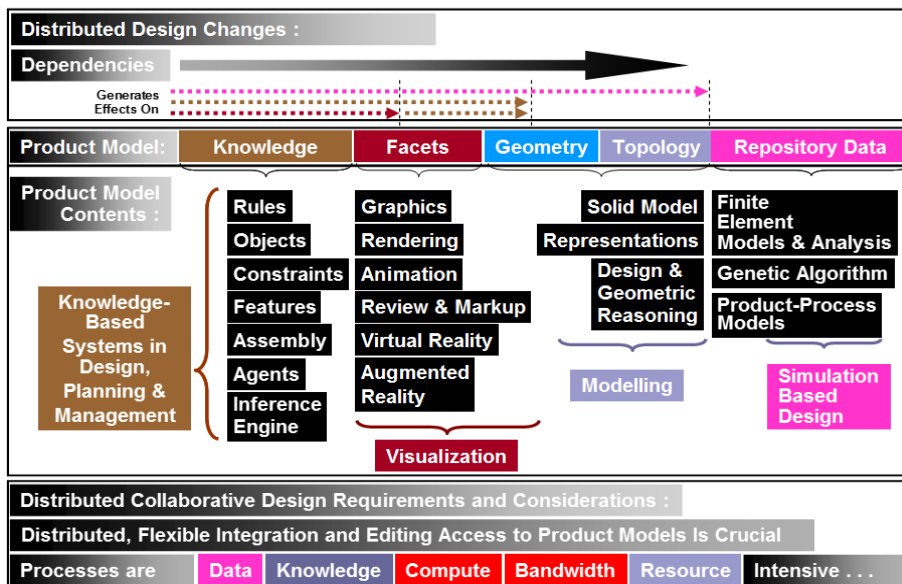


Figure 3.2: Distribution of Functionality and Data -

1.) Distributed Design Changes; 2.) Product Model Components; & 3.) Requirements and Considerations

It can be argued that the presence of these different types of functionality and data brings about the need for middleware capabilities and also increases general and flexible interoperability through managing complexity across distributed heterogeneous environments. This would be followed by a description of the architectural framework and its various elements focusing on the perspective of seamless integration and related considerations of design access and synchronization.

3.2.1 Classification and Distribution of Functionality and Data

Figure 3.2 represents a proposition involving a classification and distribution of types of functionality and data that underpin the importance of product-process modeling and distributed collaborative design functionality issues and capabilities. It proposes that within distributed environments, design changes today drive several of these functionality and data within the realm of product-process modeling and applications, require access into the product model for interaction accompanied by geometry

definition support, and generate a multitude of necessary data for product development and decision-making. Figure 3.2 also highlights the general fact that the various underlying computing processes are bound to be data, knowledge, compute and resource intensive, evidenced by the discussion on conventional CAD systems.

Looking at what would typically constitute a product model, the classified areas may suggest the important roles of utilized design knowledge within applications as far as the users' overriding objective of product design and development are concerned. An example of such design knowledge can be semantic feature modeling and yet in a more process-driven way, another would be for concept design of product configuration, and one more would be interactive fixture design, just as the feature and geometric modeling capabilities should also be considered as product design capabilities. Next, applications supported with design knowledge also require complementary and supporting technologies of interactive and flexible user interfaces fulfilled by timely visualization of product models integral to accurate design interpretation and manipulation. Product models require the support of boundary representation (B-rep) modeling (using geometry and topology) for accurate definition, solid model reasoning, and the subsequent generation of 3D facets or facet models for interactive visualization and design activity. [Shikhare 01] highlighted that engineering (facet) models can be highly complex. Without this, both product definition and access would be impossible. Last but not least, repositories refer to the persistent storage of pertinent databases and files.

Given the review and nature of conventional CAD systems, these classified functionality and data can be understood to be present altogether as a tightly

integrated standalone system. In this context, for supporting enterprise efforts in product development, the resulting product models can only be shared via databases and files hosted on dedicated server networks. However, in the first place, these repositories are really derived from the need to centralize and secure these databases and files from unauthorized access and modification.

Nonetheless, within their own right, the types of functionality and data highlighted are really integral to the exemplified application or use, as illustrated. It is also valid to emphasize additionally that these types are not only inter-dependent but they can and should be mutually exploited where necessary such as in augmenting knowledge-based and computational capabilities to the visualization of facets *per se*, given that it is firstly, really knowledge of objects that are involved in say, semantic feature reasoning, and secondly, say within an assembly/disassembly problem context, computationally efficient and accurate algorithms or libraries in collision detection are available for integration into facet-based databases to effect problem solving. What this perspective suggests is the need to consider fundamentally how these types of functionality and data are to be distributed, integrated and flexibly handled in order to be appropriate to the needs of distributed collaborative design.

Hence given today's dynamic nature of geographically dispersed product design and development activities, an applications architecture and the underlying middleware framework need to be considered. The architecture is not just based on a client-server approach; it distributes the functionality and data and highlights considerations appropriate to carrying out distributed collaborative design based on Figure 3.2. The middleware framework needs to feature seamless integration for inter-operability

through interfaces and reusability, and incorporate support for design synchronization mechanisms to collectively address the needs of distributed collaborative design.

Two design synchronization mechanisms are involved in this thesis in relation to carrying out collaborative design in distributed environments and addressing the issue of design change. As design synchronization amongst application views is vital to timely, accurate and consistent updates amongst all the users, the handling of complex 3D graphics datasets or facet models in the application architecture has to be considered [Bok *et al* 04]. In addition, when design change has to be considered in collaboration, then an appropriate means of handling design change in relation to product modeling has also to be considered in attendance with the earlier mechanism. Collectively, these two mechanisms are primary to the management of application relationships across application views supporting product-process interactions and plug-and-play problem solving methods, and thus collaborative decision-making [Mervyn *et al* 03].

This thesis focuses on these mechanisms to enable the product modeler server to support application view updates and hence application relations for collaborative decision-making. It applies to the context of design changes affecting processes such as interactive fixture design in an IPPD context. To be more complete for collaborative product development, this can be further extended to simulation-based design as fixture re-design should require new simulations to be carried out.

3.3 Applications Architecture and Computing Environment

The foundation of the computing environment for distributed collaborative design

comprises an applications architecture that follows a middleware framework. The framework is basically a layered perspective that corresponds to the classification and distribution of functionality and data. The applications architecture refers to the various client-server elements that are associated with these functionality and data, introduced and discussed in the following sequence:

1. Distributed Client-Server Architecture,
2. Geometric Modeling Server(s),
3. Product Model and Data Representation,
4. Applications Views, and
5. Support Layer of Reusable Application Classes for the Application View

These elements are discussed later in terms of the computing environment configuration and general illustrations of their roles in distributed collaborative design. These illustrations lead to the issues in design synchronization to be discussed in conjunction with present server-based product modeling system architecture using the master modeler concept.

3.3.1 Distributed Client-Server Architecture

A direct implication of using conventional CAD systems in terms of its *premier* view and *one-way architecture* is that interaction is usually only possible if the user is the only one directly working at a CAD workstation. However, requirements in distributed collaborative design and DET with regards of the Internet as a ‘flat, heterarchical structure’ lead almost inevitably to the adoption of a *client-server or more generally distributed computing* architecture, in which the server provides the participants with the indispensable resources for communication, coordination and

data consistency tools, in addition to the necessary basic modeling facilities.

Distributed collaborative design therefore requires that different participants should be appropriately provided with their own, application-specific *views* on the product model according to the activities required, *e.g.* detailed design, manufacturing or assembly planning [de Kraker *et al* 97; Hoffmann *et al* 98]. Of [Hoffmann *et al* 98,00], [Bronsvort *et al* 04] noted that no implementation of the proposed architecture has been reported so far, and that only minor design changes can be propagated back to clients as they have been proposed as CAX applications or standalone conventional systems.

Hence, a recurrent conflict in client-server systems lies in limiting the complexity of the client application and minimizing the network load, through special middleware measures highlighted earlier. In collaborative design, client complexity is determined by the modeling and interactive facilities implemented, whereas network load is a function of the kind and size of the model data being transferred to/from the clients.

A whole range of compromise solutions can be devised between the two extremes, so-called *thin clients* and *fat clients*. A pure thin-client architecture typically keeps all modeling functionality at the server, sending a passive image of its user interface to the client. This approach requires continuous image updates and screen interaction information exchange between server and clients, and creates costly network traffic. The response time would be intolerably high for many model specification actions. Whereas a pure fat client offers full local modeling and interaction facilities and maintains its own local model, communication with other clients is then required to

synchronize locally modified model data. In a collaborative environment where clients can concurrently modify local model data, preventing data inconsistencies between different clients becomes a crucial problem. In addition, fat clients place on the platform running them the heavy computing power requirements of typical CAD stations. Fat clients are typically platform dependent applications requiring complex installation procedures.

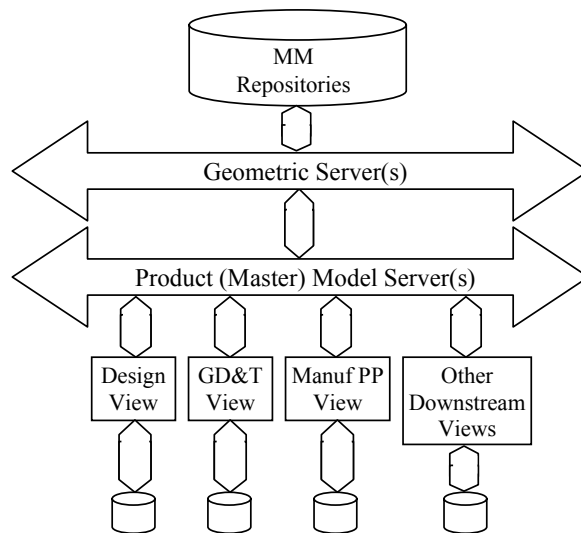


Figure 3.3: Proposed Application Architecture based on Master Modellers and Client Application Views

Following the proposition illustrated in Figure 3.2, Figure 3.3 refines Figure 2.4 in that it is based on the proposed middleware perspective of distributed geometric server(s) delivering shape definition and modification services, etc to the application views based on ‘thin’ or ‘smart’ clients. In addition, it also refines the role of the master modeler as that of supporting product data models or representations. What was proposed in [Hoffman *et al* 98, 00] which as said, relies more on conventional CAD systems as clients and does not completely address the geometric problem of handling design change and updates in a heterogeneous environment. The approach is similar to the intermediate modeler of [Martino *et al* 98]. In distributed collaborative

design, the system should still be interactive based on an appropriate distribution of functionality and data in a network context.

In conclusion, a good compromise solution can be a client-server approach, where the server coordinates the collaborative session, maintains the shared model, and provides all functionality that cannot, or should not, be implemented on the client. The clients perform operations locally as much as possible, and only high level semantic messages, and compact amounts of information necessary for updating the client data, will be sent over the network. Bearing in mind that the Internet comprises shared interconnected networks, this approach is a step to network load relatively low to allow for client interactivity at acceptable response times. An important advantage of such an architecture is that there is only one product model in the system. Clients send their modeling operations to the server, and receive feedback after any modeling operation has been performed on its central feature model, thus avoiding inconsistency between multiple versions of the same model.

Thus, a general approach (Figure 3.4) for the development of distributed collaborative applications has been undertaken such that applications can be developed independently, but easily integrated, interfaced and synchronized as set out in the architectural context of the earlier discussion. This approach, in developing distributed applications that are seamlessly integrated, is based on the perspective of a common collaborative design application 'middleware' [Schantz *et al* 00]. Early efforts in middleware development dealt mainly with connectivity issues, *i.e.* how programs on different computers can communicate with one another. These middleware technologies that deal with connectivity are referred to as distribution

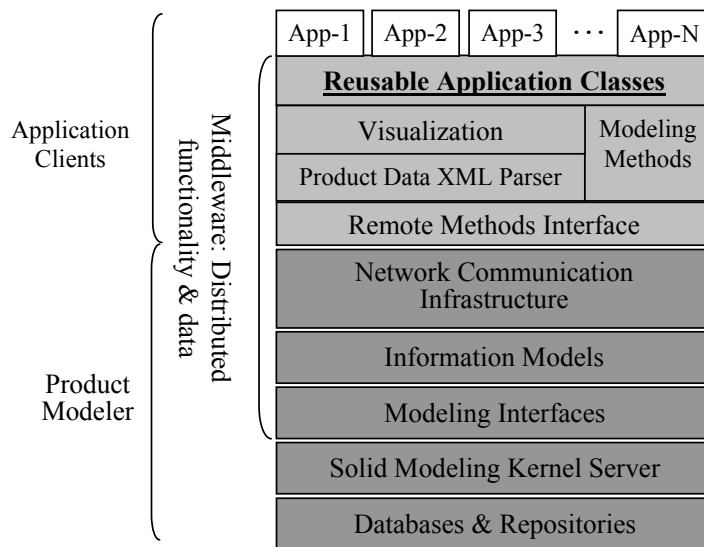


Figure 3.4: Middleware Framework – A Layered Perspective

middleware [Schantz and Schmidt, 2001]. Examples of distribution middleware include OMG’s CORBA, Sun’s Java RMI and Microsoft’s DCOM.

Distribution middleware technologies are at a matured stage today. It can also be argued that the Internet, as a network of networks, has been made possible through the most basic of all middleware, *i.e.* Transmission Control Protocol/Internet Protocol (TCP/IP), the core transport and network protocols for creating connections (such as sockets) between IP addresses on computers, moving data packets between these connections and ensuring the correct delivery of such data. TCP supports many of the Internet's most popular application protocols and resulting applications, including the [World Wide Web](#), (WWW), [e-mail](#) and [Secure Shell](#). The Web was made possible with higher level protocols such as the Hyper Text Transfer Protocol (HTTP) and the Uniform Resource Locator (URL), and represents a significant information evolution from text-based display and presentation to inter-connected hypertext media-rich display and presentation allowing for widespread adoption, without which search

engines, portals and information channeling protocols such as Really Simple Syndication (RSS) cannot be built upon, to name a few.

The viewpoint here is that middleware infrastructure technologies have since progressed to dealing with other context-based and domain-specific issues in developing practical distributed systems requiring specific functionality and data issues to be addressed. From such a perspective, this thesis had once started with the proposition that not only basic middleware is essential but also context-specific domain-driven middleware framework and capabilities would be necessary to the development of distributed collaborative product and process design systems integrating disparate designers and engineers in a distributed collaborative design context spurred by product design activities and driving processes such as in interactive fixture design. Providing direct access to product and process models, distributed collaborative design requirements demand more capabilities to accompany collaborative product feature reviews that leverage web and Internet constructs to carry out team discussions [Huang and Mak 03].

Basically, in the context of distributed collaborative design, middleware is thus a set of layers or multi-tiers of software interfaces and components that sit between solid modeling kernels and manufacturing applications (Figure 3.4). The layers are distributed between application clients and a central server and can be seen also as a set of services. The solid modeler interface and information model layers of the middleware are part of the server, while the reusable application classes are part of a client. The communications infrastructure interfaces clients and the server.

The solid modeler interface is responsible for interfacing the server end with solid modeling kernels. The information model layer contains information from the solid modeler in a neutral form. It could also contain other information deposited by application clients. The communications infrastructure allows applications to technically make remote function calls to the server. The reusable application classes are a group of reusable classes that applications use for development.

3.3.2 Geometric Modelling Server

With the shortcomings of conventional CAD systems as well as the above considerations, it is necessary to engage a geometric modeling server to provide remote and central access to editable standard B-reps, to maintain persistency in the product model, and to support design changes through shape modifications that change the B-rep, the detection of which will be salient to design synchronization. Another reason for a geometry modeling server instead of a CAD system acting as a server is the high computational costs associated with re-executing the whole model history in such a system, other than its monolithic and complex state. Furthermore, many conventional applications in CAD/CAPP/CAM/CAE rely on geometric modeling kernels. What this can mean is that as long as a geometric modeling kernel can be reliably used to address the concerns of the user in product design, development and even exchange, the operational knowledge-based and application-based aspects of product modeling and development become pertinent and feasible.

As long as a geometric modeling server is running, there is no need for ‘geometry certificates’ to provide for the associative relations between geometric faces of the

product model that has to appear on different standalone CAD systems [Hoffman *et al* 98, 00].

3.3.3 Product Model and Data Representation

In general, product databases are an important consideration given that they provide the basic repository means of storing information persistently. In the sense that geometric or CAD models need to be generated, various design and manufacture domains contribute over time, vital engineering information and analysis data which need to be stored in databases. Thus, as a result, the term ‘master model’ is frequently used to describe the total product model.

In the context of distributed collaborative design involving heterogeneous environments and diverse practices, it can be said that a neutral 3D product data representation is needed to ensure flexible and integrated access to a product modeling server. In particular, the suitable information model implementation as a data representation is a concern. These are valid concerns in the context of keeping the product or master consistent as various design changes resulting from different design/manufacture domains, occur. This is even more valid as the concept of application and feature view requires an explicit perspective of the product information model coupled with the necessary interaction and manipulation capabilities for access to design and editing.

Unlike say, semantic feature modeling with view support on the one hand and the lack of middleware representation and development in the context of [Hoffman *et al* 98, 00], the approach here proposes a basic product information model comprising basic

geometric shape elements and entities that could be sufficient for interactive fixture design, say, to take place since it can be considered that such process-based domains do not inherently need design editing capabilities. The use of geometric elements can be extended to create relationships with other applications in order to propagate changes for synchronization. Unlike features which are application dependent due to its semantics, geometric shape elements are primary and generic to a wide range of applications. Many downstream application problem solving can thus be supported based on such geometric elements, and hence, providing for relationships with geometric elements through this type of product data representation is meaningful in the context of handling design change in general.

Such a data representation in principle can thus act at the core of a flexible, multi-perspective product modeler whereby the perspective is application-specific. A good instantiation of this principle is both the dependence of interactive fixture design on this data representation and yet, the extension into fixture design configuration as augmentations.

3.3.4 Application View

The presentation and processing of relevant information from the product or master model is called a *view*. The importance of supporting *views* in the distributed collaborative design context of multiple users and domains was noted [de Kraker *et al* 95] [Hoffman *et al* 98, 00].

In particular, the concept of *views* arises to resolve or circumvent the *premier* design view constraint of conventional systems. The paradigm of a resident *premier* view

means the design view must be used to make all net shape changes. From such a design view, all other views arise as derivative. Thus, commercial CAD systems cannot accommodate such view differences [Hoffman *et al* 98, 00]. *Views* are dependent on how the information modeling is handled from the product or master model in ways pertinent to the application (typically a process method). They thus comprise product-process interactions that require accuracy and consistency through synchronization. Such interactions may arise from the private or proprietary methods of users or companies that handle the view.

The middleware perspective of providing a common geometric modelling service for applications to create and access data really brings about the possibility of coordinating multiple users as they manipulate geometric data and information for product and process modelling. This has the important ramification that the client applications can interact and inter-operate simply because of their application or context specific views containing their requisite customisable and extensible functionality. This is of course due to the role of object-oriented Java-based classes for visualization, manipulation and the respective application functions.

3.3.5 Reusable client classes for application views

Design and manufacturing planning are highly multi- and inter-disciplinary, and must be dealt with in a domain-specific manner. Thus, the fundamental technical requirement is that software client classes must be reusable and extensible to include domain-specific application-bound functionality without imposing additional interfacing burden.

These common classes can be used as the basis for developing applications and hence aid to reduce the time and cost of developing applications. In the present system, three common classes have been developed: (i) remote interface class (ii) product data XML parser class and (iii) Visualization class. Collectively, given the augmented Product data representation, these define the present application *view* that is distinct from conventional CAD systems, allowing for integration of specific domain methods and support for private and proprietary information.

The client remote interface class has been defined to facilitate interaction with the server implementation of the interface. This allows applications to call the modeling functions of the modeling kernel as required through the solid modeling interface. In principle, this solid modeling interface can be further interfaced with other modeling kernels through adaptation such as ACIS and OpenCASCADE.

3.4 Distributed Collaborative Design and Design Synchronization

In Figure 3.5, five relevant roles and application views can be generally illustrated:

- (a) Part Design capability to currently engage Fixture Design. This can also eventually support product design.
- (b) Fixture Design capability to carry out a vital tooling process to plan for a manufacturing process, *i.e.* Numerical Control-aided machining.
- (c) Numerical Control-aided machining planning for tool path generation.
- (d) Simulation-related Modeling to prepare fixture design for manufacturability assessment, *i.e.* fixture analysis.
- (e) Simulation capability to execute and monitor the fixture analysis job(s) on a compute server(s).

- (f) Analysis capability to interpret and assess the results of the analysis job(s).

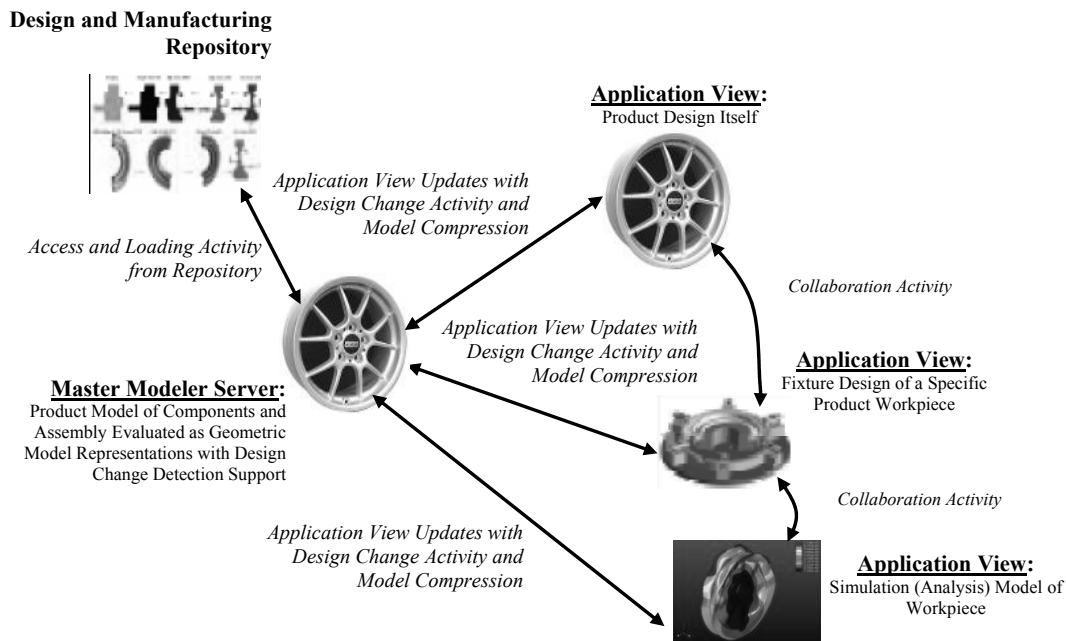


Figure 3.5: Product Modeling in Distributed Environments - Application Views & Relationships with Relevant Design Synchronization Support for the Example of a Forged Car Rim

Assuming this is a conventional environment comprising distributed standalone CAD systems; it would be valid to highlight several drawbacks with Chapter 2 in mind. These include the massive amounts of CAD data exchange taking place between the users, proliferation of CAD data models without being able to track a single master product model with consistent boundary representations, and inability to handle design changes and hence synchronization toward application updates and collaborative decision making.

With Figure 3.3 proposing the application architecture, Figure 3.5 should then depict the different roles of design and engineering activities as architectural elements, *i.e.* application views and product modeler characterized as distributed environments. The

appropriate support needed to enable this requires a distributed collaborative design computing environment with a middleware framework to support seamless integration and design synchronization mechanisms.

For this purpose, Figure 3.5 further illustrates the scenario whereby fundamentally design change activity is depicted and application view updates need to take place in a coordinated manner. In this regard, design synchronization across a distributed environment is depicted by the need for model compression and proper design change detection at the product modeler supporting relationship management for collaboration decision-making in product-to-process and process-to-process interactions. This scenario expresses an instance of the interacting role of the repository and the master model with geometric server(s) towards relevant application views involving product design, fixture design or planning, and simulation.

Relevant issues concern not just having distributed application views interacting with the product modeler and its product data representations. They are also about how each application view can perform its tasks across distributed environments with appropriate middleware support in view of the nature of heterogeneous environments, compatibility issues and the shared bandwidth and expected latency of the Internet. Such support extends to the need for design synchronization mechanisms with regards to timely, accurate and notification updates due to design changes as the routine nature of rapid collaborative product design and development.

The author notes that the Applications Relationship Manager (ARM) is also an extension from the system's reusable client classes to support design synchronization

in an IPPD context involving design changes affecting fixture planning with the goal of seeking adaptive rapid solution responses [Mervyn *et al* 03b, c].

Basically, the Applications Relationship Manager allows applications to build light weight or informational relationships with the product model at the product data representation level of geometric shape elements, rather than any feature-based scheme. Synchronization of product-process model informational relationships among all applications is then carried out through the Applications Relationship Manager. Although the current framework and deployment of application architecture proposes the use of geometric modelling services driven by application relationship management, several important aspects can be highlighted. It is presently a one-way (or single directional) architecture for product-to-process interactions excluding say, process-to-process interactions, for instance fixture design and simulation. It does not support design editing functions for shape modification using solid modelling operations (as the primary cause of design change), other than basic modelling functions. It is hence also not integrated together with synchronization mechanism of model compression for handling application view updates based on design change detection requiring complete capture of B-rep changes during shape modification so as to generate the relevant geometric shape entities for driving model compression and updating product data.

These considerations require the incorporation of primary design synchronization mechanisms into the middleware framework such as integrated and incremental model compression with design change detection particularly at the geometry and topological levels. In future, an overall multi-way collaboration architecture is

preferred to exercise bi-directional associations between interacting design and domain-specific processes for greater collaboration flexibility amongst application views.

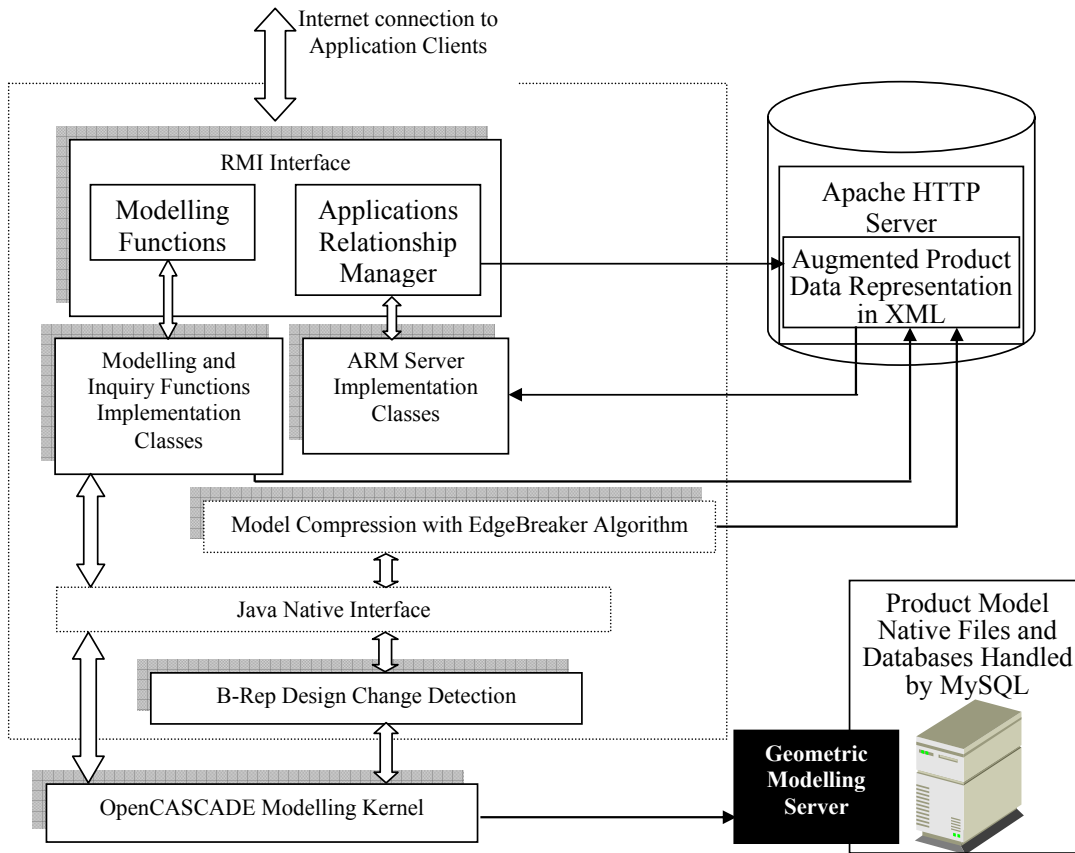


Figure 3.6: Product Modeler Architecture

Figure 3.6 hence shows the present product modeler architecture based on the middleware framework with the characteristics of inter-operability, modularity and reusability. This reveals the primary product modeling capabilities supporting the abovementioned architectural elements, enabled with compression and design change detection interacting with an augmented product data representation. A number of interfaces have been implemented in the present product modeling server architecture: Modeling and fixture design functions [Ratnapu01] [Senthil kumar *et al* 01], ARM

implementation [Mervyn *et al* 03], and the leveraging of the model compression technique [Bok *et al* 04] and design change detection. The basic modeling function interface allows application clients to make remote calls to a solid modeling kernel, giving application clients the ability to manipulate and interrogate the product model. This allows the development of applications without the installation of a modeling kernel on every machine the application is to be run.

When design change occurs to a product or workpiece, a distributed application view of an interactive fixture design process should be updated with the appropriate incremental view updates together with the relationship management of modified or deleted shape elements provided by design change detection. Once this can occur, it is correct to indicate that the application view and process should be able to employ its problem solving methods to update or re-design changes or variables affected by the design change using the augmented product data representation.

In this context, a problem solving methodology or logic, realized in a process application view such as interactive fixture design, is used to demonstrate how design change updates and fixture re-design work together through the augmented product data representation. The appropriate and careful consideration of view update capability has been based on the concept of distributed data and functionality expressed in Figure 3.2. It requires an analysis of algorithms for dealing with 3D facet datasets or models with an understanding of requirements of view integrity for design and design change. The incorporation of relevant algorithms affecting facet datasets or facet models, together with an algorithm for design change detection, is presented in Chapter 5 in the context of enabling design synchronization mechanisms, evident at

the bottom more fundamental part of the product modeler architecture.

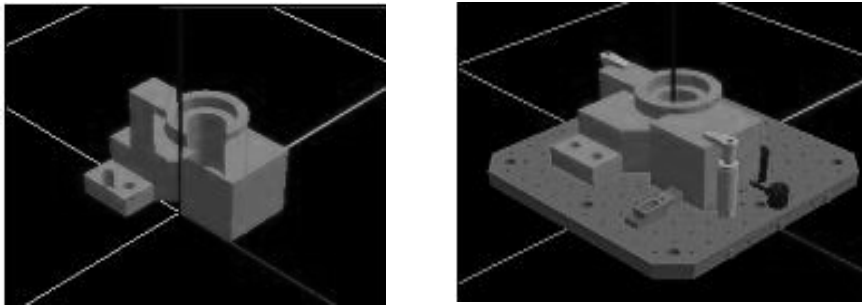


Figure 3.7: Workpiece Design and Corresponding Fixture Design

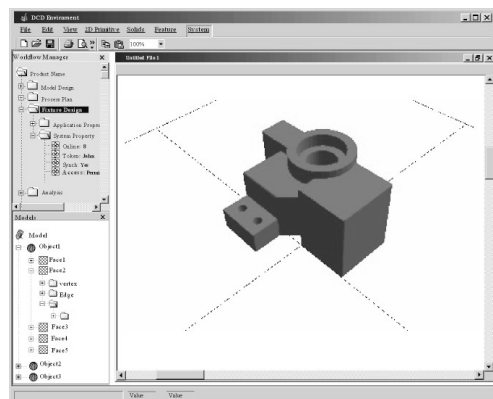


Figure 3.8: Design Application View

Figure 3.7 shows how a single interactive fixture design system has been developed and enabled with the architectural framework with a viewing of a workpiece. Figure 3.8 shows an implementation of a design application view of the workpiece in a distributed collaborative system environment. This features the development and feasibility of the middleware framework and application architecture, as will be presented in the following chapter.

3.5 Discussion and Summary

An application architecture requires a middleware framework. It comprises architectural elements corresponding to the classification and distribution of

functionality and data relevant to the field of distributed collaborative design. These elements together with the associated capabilities and techniques in information and product modeling form a distributed product modeling architecture that allows clients to concentrate on applications whilst the product modeling itself is supported by a geometric modeling server. Relevant geometric kernel modeling systems all have boundary representation capabilities together with interfaces for most solid modeling operations to be undertaken. However, these should not cause interoperability problems and in particular, impede the need to collaborate especially during design change, given the key issues of distributed collaborative design.

The application view is hence an important client and domain-specific view of the product design and development enterprise. It needs to be separate from the geometric kernel system which is then best to be an Open Source to mainly free up the concerns of users and support design change issues such as face tags and other shape entity processing and association with applications.

The application view then can be best supported by the client's application (domain) such as interactive fixture design, and it should also be responsive to design change which needs to be detected at the geometric modeler server and handled for relationship management, view accuracy and consistency for propagation across distributed environments. The following chapter provides the key developments of the computing environment demonstrating distributed interactive fixture design. This environment shows that the underlying middleware framework and application architecture design involving fixture design problem solving logic, *i.e.* rules and product data representation can be harnessed from product modeler to application

view in a distribute environment. In connection with this, the client remote interface has been extended to allow associations to be created between the product model and fixture design for managing product-process inter-dependencies [Mervyn *et al* 03b]. The associations further allow different types of relationships to be set up and activated at the face tag level by the client application. To support the client, the visualization class is responsible for view updates through visualization of the product models.

The underlying middleware framework, however, needs to incorporate design synchronization to support design change and application relations for collaborative decision-making. In particular, it is not helpful if the product or workpiece has to be re-imported into the product modeler server when design changes are occurring. Such re-importing basically causes a new boundary representation to be re-emitted leading to the loss of consistent shape entity identities or tags. As well, such re-importing or even handling of design change updates implies complete re-computation of visualization information and data such as 3D faceted models with their associated complexity. Therefore there is a need to explore techniques to handle 3D faceted models to expedite application view update and as well, during design change. In this way, the maintenance of application relationships for product-process interactions and collaborative decision-making would be more integral. These design synchronization capabilities are thus covered in Chapter 5 as necessary middleware mechanisms for distributed collaborative design involving design change.

Chapter 4

Framework Development and Interactive Fixture Design Application in Distributed Collaborative Design

This chapter has the objective of presenting the design and development of the computing environment consisting of the framework's necessary architectural elements to support distributed collaborative design in the application context of interactive fixture design. A middleware perspective of distributed functionality and data appropriate to distributed environments has been necessary. This has led to reusable architectural elements that can be appropriately deployed in distributed environments. The development of these elements is exemplified by a description of how interactive fixture design as an application is integrated and supported. The emphasis is on how these architectural elements support distributed collaborative design across heterogeneous distributed environments which will later require considerations toward the goal of design synchronization support with a view towards design change detection, *i.e.* the mechanisms for timely, accurate and consistent updates of product model and application view.

4.1 System Architecture and Overview

The middleware framework for distributed collaborative design is required within an overall distributed computing. It requires a seamless client-server architecture

computing environment (Figure 4.1) appropriate to how functionality and data can be conceptualized for distribution (Figure 3.2) and how such a distribution reflects a layered middleware perspective (Figure 3.4). It is elaborated here in terms of its architectural elements and their implementation choices, inter-relationships and support for distributed collaborative design.

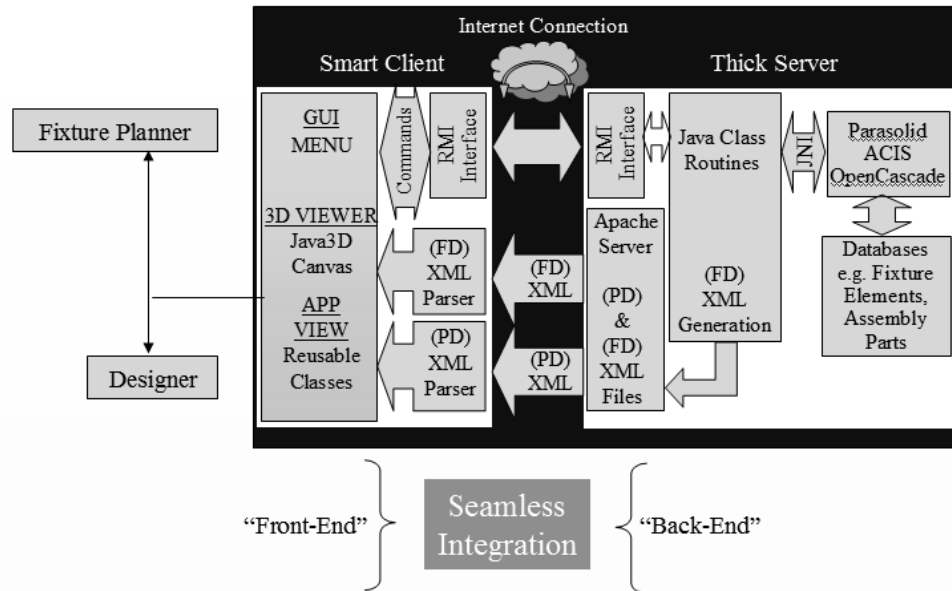


Figure 4.1: System Architecture for Interactive Fixture Design

The proposed approach of distributed client-server computing has been demonstrated in the case of an Internet-enabled interactive fixture design system [Senthil kumar *et al* 00] [Mervyn *et al* 03a, b] for distributed design based on the use of reusable client classes in an integrated Internet-enabled design pilot system [Ratnapu 01]. [Mervyn *et al* 03a, b] subsequently introduced the idea of depositing change information based on [Hoffman *et al* 00] to demonstrate adaptive fixture process re-design based on the use of genetic algorithms. It has been noted earlier that such design change handling is informational or lightweight. Complementing this, examples of data-oriented or ‘heavyweight’ design change handling would be the area of semantic feature

modeling supporting improved design intent beyond form features, and the proposition of improved design synchronization through use of lossless model compression techniques for timely, accurate and consistent application view update.

Briefly, this client-server architecture has a clear fundamental distinction between the client's and the server's share of program execution. Typically, the client end has to be first downloaded and then installed on the user's machine. The client then sets up the Graphical User Interface (GUI), which contains menu items for different functions and provides the 3D work area for visualization and manipulation. The server end of the system basically contains all the implementation of the modeling functionality, which involves creation and manipulation of CAD models. It is made up of a few distinct parts. These include the solid modeling interface routines, the solid modeling kernel itself and an Apache HTTP server providing a convenient data transfer protocol. The server end processes the requests from the client and returns results in appropriate forms depending on the kind of functionality requested by the client. It is extensible with modeling functionalities implemented using Java routines that call on the solid modeling API. Java's Remote Invocation Method (RMI) is used to facilitate the interface between client-side and server-side routines. The actual transfer of data for design communication is carried out in the form of an Extensible Markup Language (XML) schema.

4.2 Application View

The current framework implementation of the application client view has adopted Java and Java3D to fulfil the role of the application view in 2 crucial aspects: application development and visualization. The overall approach is to employ the

Java programming language for its flexible object-oriented approach to software development especially in terms of portability and its API libraries supporting distributed computing.

4.2.1 Visualization

One of these APIs is Java3D which employs 3D scene graph programming techniques to spatially organize objects and handle 3D object scene rendering, visual navigation and selection of these objects. This is integrated into the development of the presentation of user-interactive capabilities (as in the widgets and mouse operations behind the Graphical User Interface (GUI)); and the application view-specific functionality such as algorithms and rules in the form of reusable application classes.

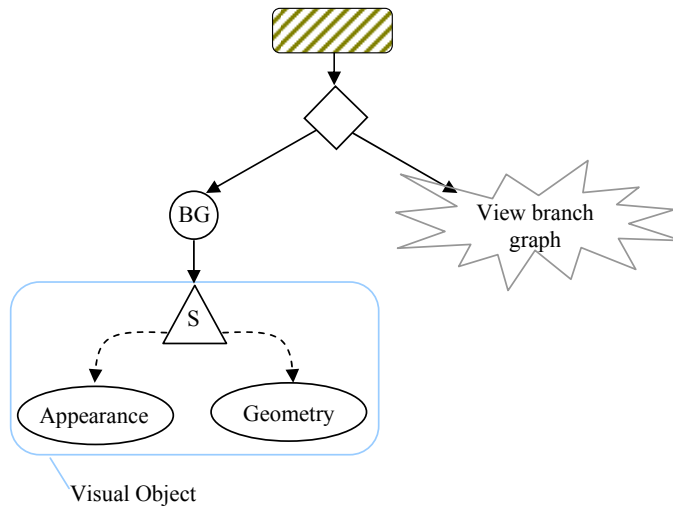


Figure 4.2: A Shape3D Visual Object(s) inside a Java3D Scene Graph

Figures 4.2 and 4.3 illustrate Java3D scene graph technology. Figure 4.4 further illustrates an instantiated scene graph that must be coupled with a Java3D canvas for presentation and visualization of the object space. The Java3D canvas is fundamentally an image plane or space, on which a rectangular array of pixel content is projected to form the rendered image on the screen for a viewpoint (Figure 4.5).

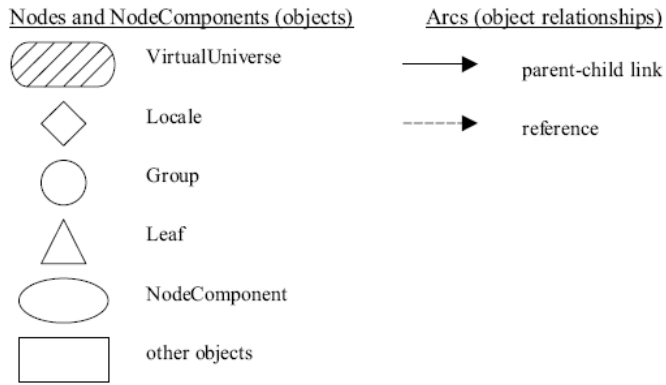


Figure 4.3: Symbols Used in Representing Java3D Scene Graph

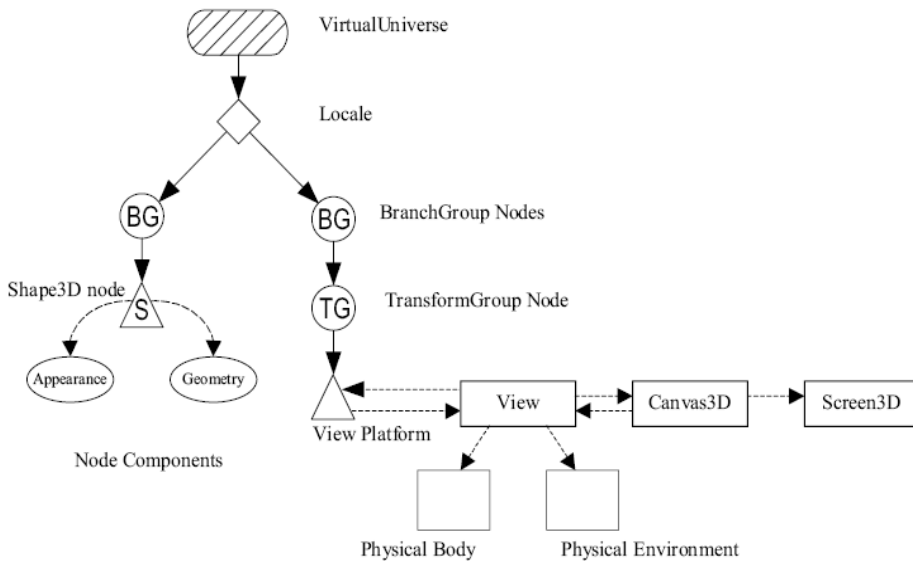


Figure 4.4: A Java3D Scene Graph Integrating Scene Graph's Object Space with a View/Screen Canvas

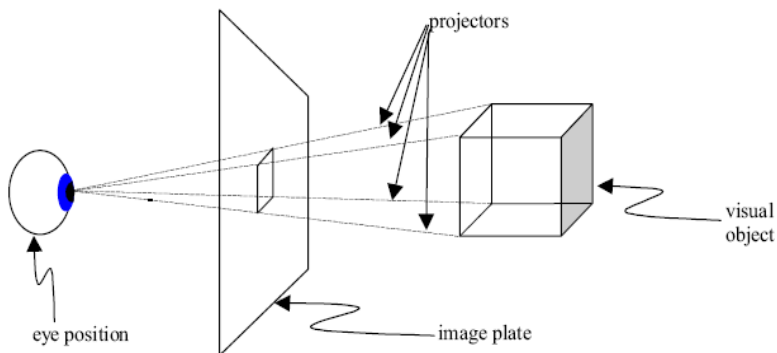


Figure 4.5: Rendering Object Space on Image Plane in a Virtual Universe

Java3D is designed to provide simple and flexible mechanisms for representing and rendering potentially complex 3-D environments. Like Java, it offers advantages of application portability, hardware independence and performance scalability. The scene graph contains a complete description of the entire scene, or virtual universe. This includes the geometric data, the attribute information, and the viewing information needed to render the scene from a particular point of view.

The geometry data presented through the perspective of a product model consisting of essential boundary representation information, is parsed and used to construct a dynamic Java3D scene graph. The scene graph needs to be carefully constructed as frequent traversal through the tree is required during further interaction with the 3-D shapes by the user.

The Java3D API's scene graph-based programming model provides high level language constructs for creating and interacting with 3-D geometry and tools for constructing the structures used in rendering that geometry to support the application view. Interaction is provided by means of visual behaviors or Java3D methods such as:

- i. View Navigation through Mouse button control – Objects in the scene graph can be rotated in all directions by using the left mouse button and viewed.
- ii. Zoom – Objects can be zoomed in and out using the right mouse button.
- iii. Picking – Objects can be treated as picking entities with different contexts.

Java3D therefore makes it simpler to construct and manipulate 3D object scene environments as Java objects compared to a more basic but powerful approach to 3D

graphics programming known as display lists which would require the programmer to implement more scene constraints and visual controls. A standard tool for this would have been the OpenGL programming language.

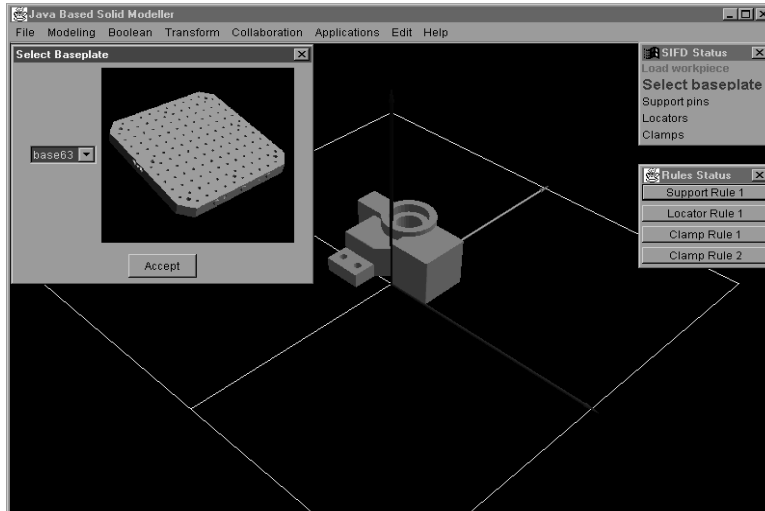


Figure 4.6: Application View with Java3D Canvas for Interactive Fixture Design

Specifically, the application view, as shown in Figure 4.6, therefore consists of a Java3D canvas and a menu bar that provides the options for starting an application, in this case, the fixture design application view and process, at the client end.

4.2.2 Client Infrastructure

The client then executes its share of program execution beyond setting up the GUI and the canvas for rendering based on an event-based programming model to make the application view ready for the user interaction. For instance, upon a modeling request from the user through menu interaction, the client directs the request to the server over the network. It then requests and reads the results generated by the server as a result of executing the modeling requests. The results read by client from the server are processed and rendered on the canvas for visualization and interaction.

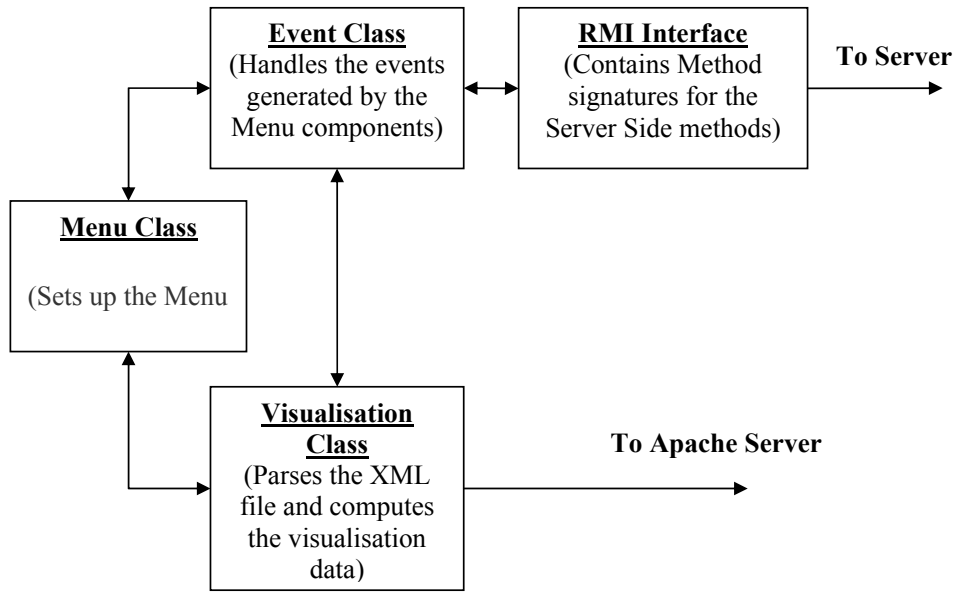


Figure 4.7: Class Architecture on Client Side

Based on this event-driven model, Figure 4.7 shows the class architecture of the application view. The Menu class starts the application. This class contains the methods for setting up Menus. Java Swing classes are essential to building the menu and the menu items. Java Swing classes are the latest set of GUI components available as a part of the Java 2 core implementation.

The Java Swing GUI components are preferred over the Java AWT components. Java AWT components are the old set of GUI components of previous versions of Java. The Swing components are preferred over AWT components because of the availability of many common GUI components which are lightweight in memory requirements.

Apart from this major advantage, Swing components themselves have more features available and provide improved interactive handling compared to AWT components.

The look and feel of the Swing components can also be controlled or tailored, even during runtime so as to suit a particular platform or environment.

The Menu Class also sets up the necessary Java3D canvas for rendering. User invocation of any of the menu items thus triggers events. These events are handled through the Event class to which the menu items delegate the necessary code to further direct the actions appropriate to each menu item in the Menu class. Such a division of code in two different classes is essentially the key to proper maintenance of code and future reusability and extensibility. Depending upon the type of event, the Event class can call either a local method or a remote method on the server side in a distributed environment. The actual implementation code is thus suitably decoupled by existing in those methods. For any remote methods (methods on server side), only the methods listed in the RMI Interface could be called. In other words, it is possible to provide in future support for different application process and view requirement whilst yet maintaining a common set, an approach essential to the framework conceptualization and development.

4.2.3 Application View Visualization Functionality

Application view visualization is core to product design and requires the geometric definition of the product model to drive it. The Visualization class thus handles the rendering of models in the canvas once it is set up by the Menu class.

More importantly as distributed collaborative design is not about visual simulation which can be compromised in quality, the visualization data needs to be remotely integrated to and obtained from the server side as a result of executing a remote

method on the geometric modeling server. This execution is typically known as tessellation and results in numerous facet data.

The geometric definition is obtained from this modeling server during runtime captured in the form of an XML schema. It is an integral part of product information representation and modeling to be subsequently presented as part of a design synchronization mechanism. The XML schema is elaborated in an XML file that is to be parsed and interpreted by the Visualization class to set up the Java3D scene graph objects for rendering and interaction. In Java3D, a visual object can be defined using just a Shape3D object and a Geometry node component. Optionally, the Shape3D object can also refer to an Appearance node component that assigns it additional properties for display effects. In addition, due to the information captured in the product information model via XML, additional and necessary interaction capabilities such as picking edge and faces are added. It has to be noted that visualization is vital for accurate interpretation and manipulation to support the application in context. Poor or low-resolution tessellation from the geometric modeling server would not be appropriate.

To clarify on the distributed collaborative design context, it is impractical in the conceptualization of a framework for interactive design collaboration to occur across distributed environments if one were to only conceive a remote display capability akin to a shared whiteboard whereby the client terminal merely accesses the application and there is no computational process or algorithm of any kind. The client terminal therefore technically is only responsible for displaying pixels that are dynamically generated by and refreshed from the remote application server, onto its own

rectangular image plane. Additionally, for visual interaction with the application, all the key strokes and the mouse movements and events are just passed over the network to the remote machine. The remote server processes these events and generates the display data which is passed across the network to the client machine. Evidently, for argument sake, this approach requires huge amounts of data transfer across the network taxing on shared bandwidth.

As a corollary and observation, this in part describes better what has recently become commonplace in distributed environments – real time video streaming. This capability is suitable since video content is crucially standardized in data format and resolution, widely adopted through established standards, and intrinsically requires little active interaction and ultimately no editing as it is already ‘authored’ and managed into huge but static repositories. In conclusion, such a plain distribution of functionality and data across a distributed environment is clearly more pertinent in the form of multimedia-oriented functions. In comparison, in distributed collaborative design, more specific or peculiar approaches and mechanisms are required to facilitate the needs of product design.

4.3 Server Infrastructure and Geometric Modelling Services

Distributed environments necessitate a middleware framework and layered perspective that has domain specific is compatible with the needs of distributed collaborative design in supporting interacting product and process design applications. As highlighted earlier, this approach, in developing distributed applications that are seamlessly integrated, is based on the perspective of a common collaborative design application ‘middleware’ infrastructure [Schantz *et al* 00].

4.3.1 Server Infrastructure

In the domain context here, the middleware is a set of layers of software components that sit between solid modeling kernels and manufacturing applications (Figure 3.4). The layers are distributed between application clients and a central server. Here, the solid modeler interface and information model layers of the middleware are considered a fundamental part of the server, while the reusable application classes are part of a client, allowing for specific application view development.

To enable communications across middleware layers in general, the infrastructure is based on using the Java RMI approach. Requests from the client to server for the modeling operations for instance, have been developed using Java RMI. Java RMI is a high level communication capability between Java objects over the network. It allows one to use the methods of remote Java objects as if they were locally available on the same computer. Java RMI has been preferred over other mechanisms like CORBA (Common Object Request Broker Architecture) due to the middleware nature of applications in seamless distributed environments required. CORBA is more general as it has been targeted at communications between different objects of different legacy languages and platforms whereas Java RMI is only applicable to communication between Java Objects.

Since Java by its very nature is platform independent through the technology of virtual machines, Java RMI allows Java objects across different platforms to communicate with each other by transmitting data parameters and executing functional code, thus providing interoperability. The only feature that Java RMI lacks

in respect to CORBA is direct communication between objects of different native programming languages. However, this can also be circumvented using Java Native Interface (JNI) and Java RMI together. JNI provides mechanism for integrating Java with other languages and RMI can use these Java interfaces to manifest abstractions or methods to cater for communication with remote Java objects.

For Java RMI, all the remote methods (methods that will be accessed remotely, in general, either by the client or the server) need to be declared through a set of declarative interfaces. The communication actually takes place between these declarative interfaces which generate stubs (which are Java classes containing method signatures of subsequent implementations of the corresponding remote method). These stubs are then dynamically shared across all participating computers in a distributed environment ready to be invoked.

In summary, those modeling requests activated through the menu on the client GUI generate the calls to remote methods on the server side. Java RMI then listens to these calls transparently and directs them to appropriate classes for further action, in this context, on the geometric modeling kernel.

4.3.2 Geometric Modelling Services

Geometric modelling services are fundamental and refer to the ability to create and manipulate geometric models, and access geometric data that are core to solid modelling and product definition. This is proposed as a generic middleware service for three reasons. Firstly, it is observed that many of the currently developed CAD, CAPP and CAM applications do not develop their own geometric modelling

capabilities, as noted in earlier Chapters given the tremendous complexity of developing kernels as well as their intrinsic characteristics underlying conventional systems. Thus, many of these applications in CAD/CAPP/CAE/CAM are developed based on either third party, proprietary or even Open Source geometric modelling kernels. These geometric modelling kernels provide functionality to build, manipulate, view and inquire geometric models during runtime. Examples of applications developed on third party geometric modelling kernels include SolidWorks 3D CAD software from SolidWorks Corporation, ESPRIT CAM software from DP Technology Corp and PATRAN Finite Element pre-processor from MSC. Given the need to share and interoperate product models, it is therefore sensible to provide geometric modelling services as a common service.

Secondly, many applications use geometric modelling kernels only to extract necessary information from product data to carry out their own tasks. Providing the ability to access geometric data from a common service would remove the reliance of these applications on geometric modelling kernels just for information extraction. This is supposed to have been the motivation for product standards to share both product definition and specific process and task information, notwithstanding the challenges of distributed collaborative design and early design change. However, it has been noted in Chapter 2 that many applications or systems related to product design, development and simulation are not suitably integrated for distributed collaborative design, notwithstanding the important concerns of conventional CAD systems.

Thirdly, and therefore, providing a common geometric modelling service where applications create and access geometric data across distributed environments provides a unique opportunity to manage the concurrent authoring and processing of geometric data by product and process design applications.

To make use of a geometric modeling kernel, a solid modeler interface needs to be present and responsible for interfacing. Such an interface can be generically used with other geometric modeling kernels. In addition, an information model layer ‘carries’ information from the solid modeler in a neutral form and supported by the HTTP protocol for data transfer. In this context, the HTTP protocol is hosted by the Apache web server. This information model layer can also be extended to contain other information deposited by application clients, due to XML schemas.

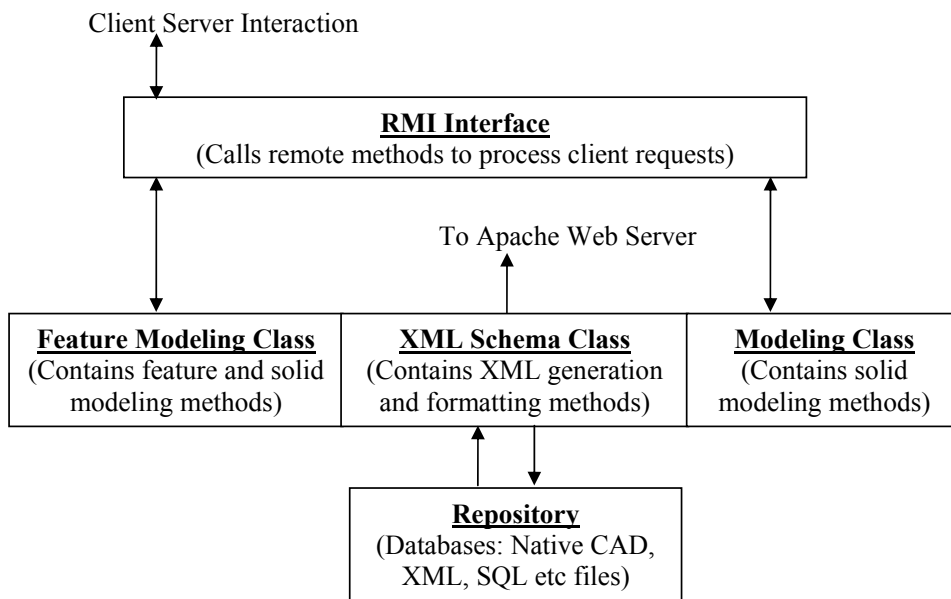


Figure 4.8: Class Architecture on Server End

In the middleware framework, the geometric modelling services are thus deployed on a central geometric modelling server based on a kernel and managed through RMI-based modelling interfaces. The geometric modelling server, shown in Figure 4.8 in the form of a class architecture, was implemented in Java and consists of the following components: (i) Java Remote Method Invocation (RMI) Interfaces (ii) Implementation Classes (iii) Java Native Interface (JNI) (iv) Parasolid Modelling Kernel and (v) Apache HTTP Server. Portability with an Open Source kernel system, OpenCASCADE, has also been achieved.

Since the implementation of the functionality provided by the server is computationally intensive and since many clients could be connecting to the server simultaneously, the server is required to possess powerful processing capabilities. As server loads can increase due to a multitude of clients potentially connecting simultaneously, a single server could in principle be replaced by a number of servers and thus simulating a parallel-processing environment using network and session management capabilities.

4.3.3 Modelling Interface and Functions

Using the RMI interface approach, modelling semantics or methods can be declared including the following:

```
public int[] createBlock(Block block)
public int[] createSphere(Sphere sphere)
public int[] createCylinder(Cylinder cylinder)
public int[] createPrism(Prism prism)
public int[] createTorus(Torus torus)
public int[] createCone(Cone cone)
public void union(int body1, int body2)
public void intersection(int body1, int body2)
public void subtraction(int body1, int body2)
public void xRotate(int bodyTag, double x, double y, double z, double angle)
```

```
public void yRotate(int bodyTag, double x, double y, double z, double angle)
public void zRotate(int bodyTag, double x, double y, double z, double angle)
public void translate(int bodyTag, double x, double y, double z)
public void scale(int bodyTag, double scaleFactor)
```

While the RMI Interface declares the methods that application clients can invoke, the actual implementation of the methods is performed by the Modelling class implementations of the involved kernel. These implementations finally make the calls into the involved geometric modelling kernel library during runtime.

In the prototype system, the Parasolid modelling kernel has been utilised to perform the operations described in these methods. As the Parasolid modelling kernel is written in the C programming language, a Java Native Interface (JNI) is needed to utilise the modelling functions of Parasolid. JNI allows Java classes to make calls to libraries written in other languages. Data of the created geometric model is then written to a Product data XML schema and hence file, and stored in the Apache HTTP server for application clients to access.

The details of the sequence of activities when any of the methods is invoked are as follows:

1. An application client invokes one of the methods of the Modelling Functions RMI interface.
2. The Modelling Functions Implementation classes invoke the necessary Parasolid functions, resulting in the creation or modification of a geometric model.
3. Parasolid (or any equivalent geometric modelling kernel) generates the necessary information to describe the geometric model based on a boundary

representation. A boundary representation basically describes a geometric model by defining the model's boundary as a set of geometric entities constituting faces, edges and vertices, and establishing dynamically its topological relations to ensure model integrity (Figure 4.9).

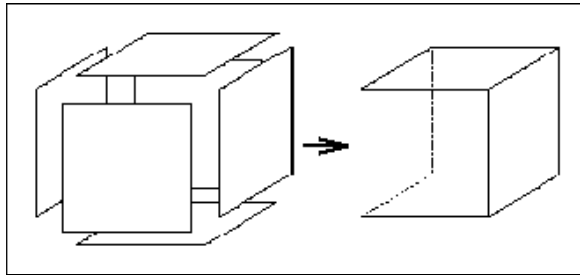


Figure 4.9: Block Represented By Its Boundary

4. The geometric model and the constituent geometric entities are identified by tags maintained in the boundary representation. As the boundary representation is internal to a CAD model, tessellation is always performed by the kernel to generate facet data to enable visualization on the application client. The modelling function implementation classes thus perform the invocation of tessellation function. The tessellated triangles can then be rendered on the application client's screen to provide a solid view of the geometric model. An example of a tessellated model is shown in Figure 4.10.

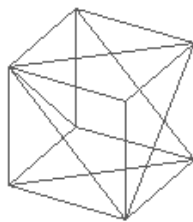


Figure 4.10: Example of a Tessellated Model

5. The information on the tessellated triangles and the geometric data entities are then written to a Geometric Data XML file as a Product Data representation and stored in the Apache HTTP server. Application clients can then access this data easily from the Apache HTTP server, visualise the geometric model and carry out further operations.

Client-server interaction consists of a client end with the functionality for setting up the GUI which contains Menu items for different modeling functionality (as reusable classes) and a Java3D canvas for rendering and visual interaction of CAD models. The implementation of the control code for the menu items and the code to generate requests to the server is also present here. However, it is clear that there is no implementation of the modeling functionality present on the client side, but instead requests are generated by the client to the server which processes the requests and returns back the results in the form of faceted data. This faceted data is rendered by the functionality available at the client, which will later on be a subject for middleware-based design synchronization to improve timely and consistent application view updates.

4.3.4 Product Modelling Server Architecture

With session management, different remote clients can thus send requests to the server for viewing and sharing of design or other information of any other remote client. The server on receiving the request from a client, retrieves the necessary information from the design database and sends it to the client. The client, with the functionality available at its end, processes and displays the data. The concept of

distributed concurrent design and engineering can be realized through this aspect based on the middleware and reusability perspective.

The data exchanged could be more than the design data. It could be product data, which includes data from different manufacturing domains. The architecture allows for exchange of product data and not just the design data alone. The visualization of the product depends on the functionality provided at the client. The architecture is extensible enough to allow for new functionality to be added to the client for visualization of any new product data, including Finite Element meshes or even high resolution faceted data organized in a cellular model for inspection.

The architecture satisfies the goals outlined in the earlier Chapters 2 and 3. By its very nature it satisfies the feature of remote access. The client gets only a little share of the total computation. The visualization and interaction capabilities are common on variety of platforms. It could be implemented by using different software tools and the choice of such tools will be discussed at the implementation stage. The architecture is also modular and extensible as new functionality can be added without having to change the existing architecture and functionality (Figure 4.11). One such extension has been the lightweight Applications Relationship Manager (ARM) which was also reviewed in Chapter 2, even though on its own, it does not explicitly handle product design changes from a design synchronization viewpoint except for information about these design changes affecting fixture planning.

The author also notes that the ARM is an extension from the use of the system's reusable client classes and RMI Interfaces to support design synchronization i.e.

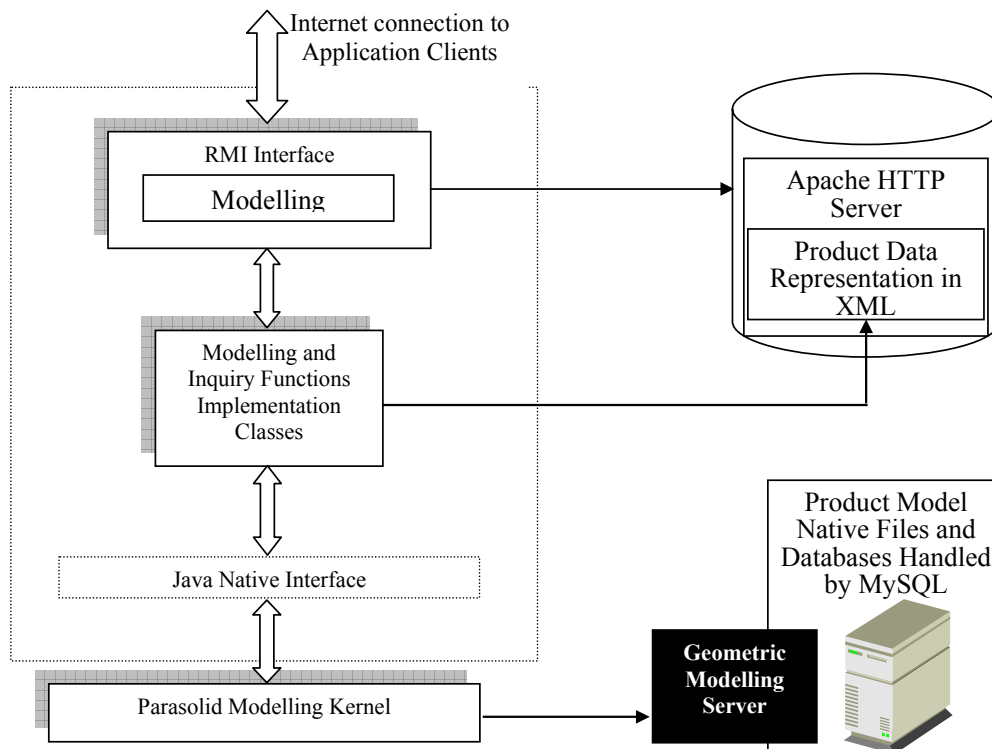


Figure 4.11: Basic Product Modeling Server Architecture

timely and consistent updates, in an IPPD context involving design changes affecting fixture planning with the goal of seeking adaptive rapid solution responses [Mervyn *et al* 03b, c]. The Applications Relationship Manager approach therefore allows applications to build important lightweight associative relationships with the product model. Further synchronization of product models among all applications is therefore to be carried out through the Applications Relationship Manager such that bi-directional associations between design and domain-specific processes can be leveraged for collaboration.

Design is highly multi and inter-disciplinary that is, domain-specific. Thus the fundamental technical requirement is that software client classes must be reusable and extensible to include domain-specific application-bound functionality without

imposing additional interfacing burden. These common classes can be used as the basis for developing techniques such as the ARM, other process applications and hence aid to reduce the time and cost of developing applications.

4.4 Product Modelling With XML

The exchanging of geometric data between client and server is carried out through XML. XML (eXtensible Markup Language) is a text-based mark-up language that has become the standard for data interchange on the Internet. Extensible Markup Language (XML) has been developed by the World Wide Web Consortium (W3C) for applications that require functionality beyond the current Hypertext Markup Language (HTML) which is specifically for web page formatting and display.

XML differs from HTML in three major respects:

1. Information providers can define new tags and attribute names to create semantics.
2. Document structures can be nested to any level of complexity and extended or inter-linked.
3. Any XML document can contain an optional description of its grammar for use by applications that need to perform structural validation.

XML has been designed to enable semantic expression and flexible implementation. Since new tags can be defined in an XML file, any data structure can be written into an XML file. In this context, the geometric data (product definition data) is written in an XML file for modeling, visualization and sharing. The hierarchical structure in the XML document is called the XML schema. The grammar defining the XML schema

is called its Document Type Definition (DTD). The DTD contains the information regarding the tags that will be used in the document. Tags in XML follow a hierarchical structure or relationship. The root tag of an XML file is always <DOCUMENT>.

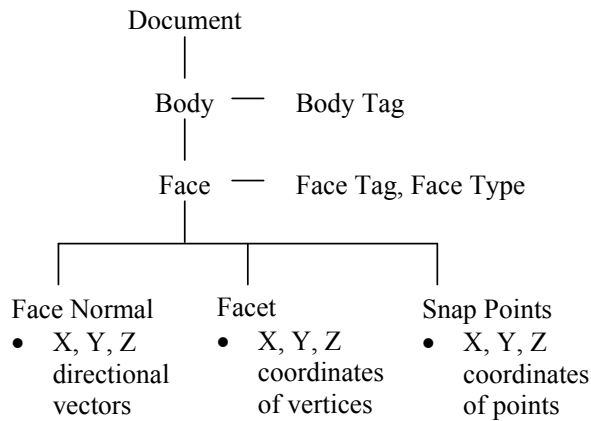


Figure 4.12: DTD Schema of Product data XML file

```

<?xml version = "1.0" ?>
<!DOCTYPE DOCUMENT [
  <!ELEMENT DOCUMENT (BODY*, FACE*)>
  <!ELEMENT FACE (FACETAG, FACETYPE, NORMAL, SNAPPOINT*,
  FACET*)>
  <!ELEMENT NORMAL (X1,Y1,Z1)>
  <!ELEMENT SNAPPOINT (X1,Y1,Z1)>
  <!ELEMENT FACET (FACETNO,X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3)>
  <!ELEMENT X1 (#PCDATA)>
  <!ELEMENT Y1 (#PCDATA)>
  <!ELEMENT Z1 (#PCDATA)>
  <!ELEMENT X2 (#PCDATA)>
  <!ELEMENT Y2 (#PCDATA)>
  <!ELEMENT Z2 (#PCDATA)>
  <!ELEMENT X3 (#PCDATA)>
]
  
```

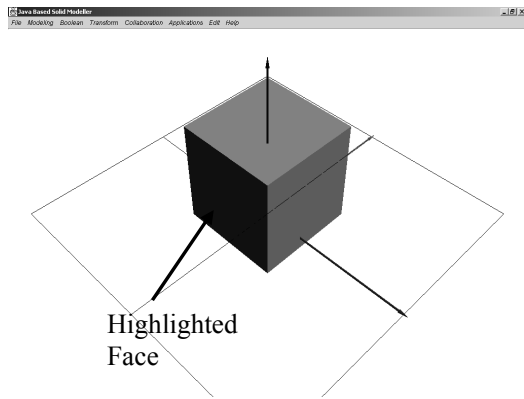
Figure 4.13: Actual DTD of the XML file of Geometric Data of a Body

To represent a product model or definition, a DTD for geometric data (Figures 4.12 and 4.13) has been proposed to demonstrate distributed interactive design and fixture planning for the Java3D-based application view. There can also be other schemas,

such as feature representations (beyond basic boundary representation), assembly structures as sets of XML documents etc, linked together. This extensibility enables sharing among different applications as this same data can be used by these different applications in different ways.

In the geometric data DTD, each body, identified by a <BODYTAG>, is divided into faces that parallel to that of the Boundary representation. A <FACETAG> is present to identify the various faces of the body, dictated by the geometric kernel. <FACETYPE> provides information on the type of the face, for example, cylindrical, plane and spherical. <SNAPPOINT> refers to the vertices of each face. These are incorporated as visual objects inside the Java3D scene graph to provide for design interaction. Each face is further divided into elemental triangles known as facets. The <FACET> tag contains the coordinates of the vertices which define each triangle or facet. Facets are the primary (standard) graphics or even geometric data resulting from tessellation of the Boundary representation of the CAD model necessary to visualization. The rationale behind such a choice will be explained in detail in the later sections in dealing with the implementation of interactive fixture design based on this middleware framework.

A simple illustration (Figure 4.14) shows a solid model of a cube and a portion of the corresponding Geometric Data DTD in an XML file. From the data, it can be seen that the <BODYTAG> of the part is 19. The highlighted face has a <FACETAG> of 150 and a <FACETYPE> of plane. The face has been divided into two facets and the corresponding vertices of the first facet can be seen in the figure.



```

<BODY>
  <BODYTAG>19</BODYTAG>
- <FACE>
  <FACETAG>150</FACETAG>
  <FACETYPE>PLANE</FACETYPE>
+ <NORMAL>
+ <SNAPPOINT>
+ <SNAPPOINT>
+ <SNAPPOINT>
+ <SNAPPOINT>
- <FACET>
  <X1>-0.25</X1>
  <Y1>0.25</Y1>
  <Z1>0.5</Z1>
  <X2>-0.25</X2>
  <Y2>-0.25</Y2>
  <Z2>0.5</Z2>
  <X3>0.25</X3>
  <Y3>-0.25</Y3>
  <Z3>0.5</Z3>
</FACET>
+ <FACET>
</FACE>

```

Figure 4.14: An Illustration of the Product Data XML

This XML file is to be read, parsed for retrieving the data contained in it. There are many XML parsers available for free on the Internet. The SAX parser as a set of Java classes available from SUN Microsystems has been used. This parser has been chosen over other ones as it conforms to the XML standard of W3C. In addition, SUN plans to implement into the core specification of Java, in which case, it will be available as part of standard Core Java classes which all Java Virtual Machines will implement.

Internally, the SAX parser parses the XML file and builds a hierarchical tree data structure called DOM (Document Object Model) during runtime. Data from this DOM can be retrieved by using the methods provided by the parser classes. The parsing is activated by the methods in the Visualization class, prior to rendering onto the Java3D canvas for the application view.

4.5 Interactive Fixture Design Application

An Internet-enabled Interactive Fixture Design (IFD) sub-system has been developed based on present middleware framework to demonstrate the feasibility of providing fixture design capabilities across distributed environments. Fixtures are devices that serve the purpose of holding the work-piece securely and maintaining a consistent relationship with respect to the tools while machining and other manufacturing operations. Capabilities include fixture rules are implemented into the application view to demonstrate the feasibility of interacting with product models and selecting modular fixture elements to create a fixture solution. These rules are aimed at making fixture designs more applicable and optimal given that original heuristics nature. Rules can also be used to guide users during design. These also rules provide the process-specific behaviour in the context of distributed collaborative design and allow for a design application view to integrate, interact and synchronize with a fixture planning application view in terms of dealing with the timely manufacturability impact of design changes downstream. This is especially plausible since there could be more than one acceptable designs are available for a given work-piece and hence the fixture design solution space could be large.

4.5.1 Fixture Design Methodology and Application Architecture

Interactive fixture design as an application view requires the application logic of fixture design in the form of rules, the design interaction and support, and access into the product modelling server and repository. This depends on the reusable application development classes, as discussed earlier, those support middleware services,

geometric model visualization and geometric databases supporting work-pieces and fixture elements, and XML-based repository data.

Interactive systems make the task of fixture design easier by constructing a fixture assembly based on input provided by the fixture designer. This assembly is also supported via an XML scheme. The inputs required by interactive systems include fixturing surfaces, fixture elements and their locations. Work in this category includes those by [Markus *et al*, 1984], [Miller and Hannam, 1985], [Nee *et al*, 1987], [Fuh *et al*, 1995] and [Rong and Li, 1997].

The aim of interactive systems is to allow flexibility to the user to arrive at detailed fixture designs for complicated parts that cannot be achieved by many of the automated systems. A limitation of most of the interactive systems lies in the fixture design sequence imposed on the user. Many systems rely on the 3-2-1 locating principle and limit the user to the choice of fixture locations based on this principle.

The interactive fixture design methodology has a typical fixture design sequence as shown in Figure 4.15, illustrated by a work-piece and corresponding fixture design (Figure 4.16). The rest of this section describes each of these activities as part of the application view for interactive fixture design.

The objective of fixture design is different from that of part or product design which involves more than access to carry out visualization and interrogation of properties and dimensions whilst the part or product design remains, in other words, fixture design on its own does not require editing to say, customize designs, which results in

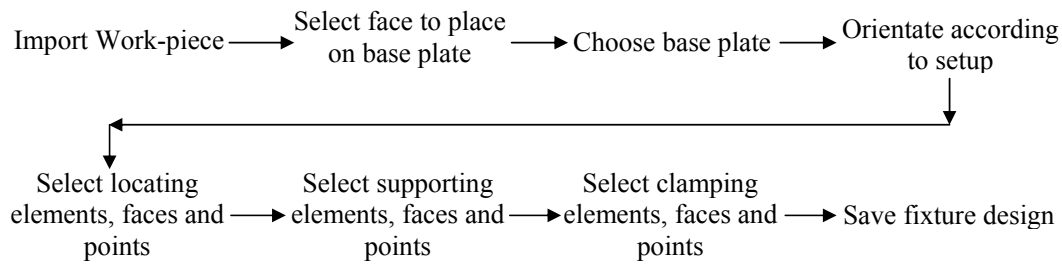


Figure 4.15: Interactive Fixture Design Sequence

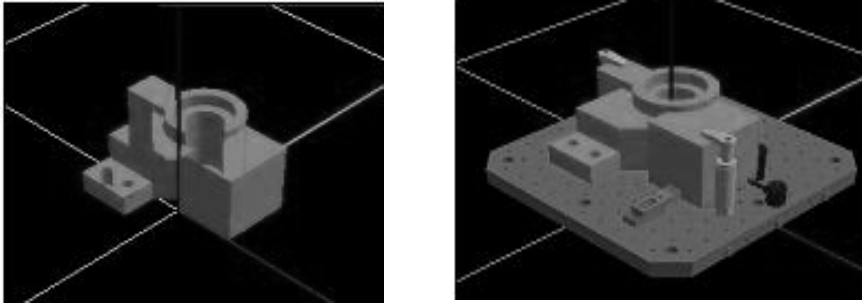


Figure 4.16: Work-piece and Corresponding Fixture Design

run-time evaluation of both the geometry and boundary representation intrinsic to the solid model and possibly history management. Thus, there can be two areas of consideration, firstly, the interaction with the visual objects within the Java3D scene graph representing the product model on the application view, and secondly, the associated behavioural actions or algorithms that are applied to these visual objects to create the functionality on the application view. The sequence in Figure 4.15 reflects essentially the first area of consideration and is elaborated subsequently, whilst the latter is facilitated by the use and extension of the reusable classes of the framework as well as the necessary Product data XML schema.

The modular fixture system used in this work is the Venlic Block Jig System from IMAO Corporation [IMAO, 2004]. The Venlic Block Jig System is a hole-based modular fixture system. In hole-based modular fixture systems, fixture elements are

fastened to the fixture base plate based on accurately positioned holes. Figure 4.17 shows an example of fixture elements fastened onto a hole-based base plate.

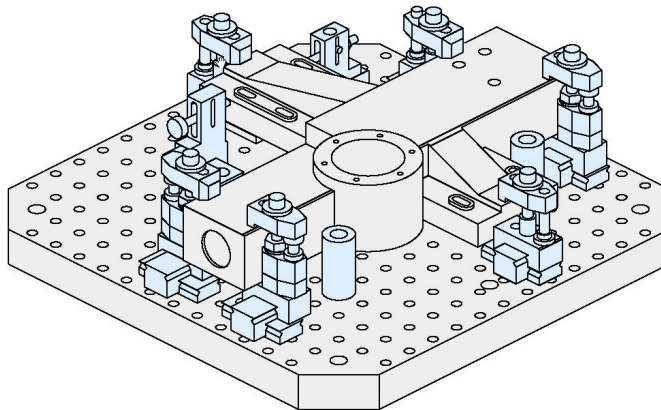


Figure 4.17: Example of a hole-based fixture base plate [IMAO, 2004]

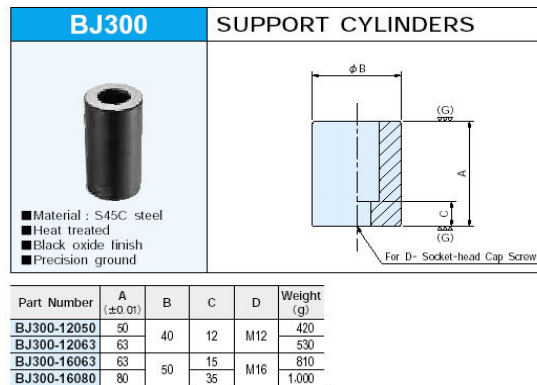
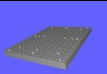
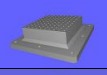
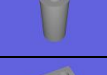



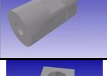



Figure 4.18 Example information stored in the fixture element database (IMAO, 2004)

The fixture element repository contains two kinds of information on fixture elements. Firstly, it contains as a repository, the Product data XML files of the different fixture elements which are otherwise conceptually completed work-pieces. These files are used to visualise the fixture elements in the application view of the interactive fixture design system. Secondly, it contains dimensional information on the fixture elements. The dimensional information of these elements is derived directly from the Venlic Block Jig System catalogue. This is more appropriately stored in a relational database, implemented using the MySQL database management system. Figure 4.18 shows an

example of the information for a supporting cylinder from the catalogue that is stored in the database. Fixture elements are organised into five groups: base plates, locating elements, supporting elements, clamping elements and adaptors, summarised currently in a database (Table4.1).

Table 4.1 Fixture Element Group Database

Group	Elements	Image
Base Plates	Rectangular Grid Plates	
	Platform Grid Plates	
Locating Elements	Locating Cylinders	
	Horizontal V Blocks	
	Round Pins	
	Diamond Pins	
	Adjustable Hex Stops	
Supporting Elements	Support Cylinders	
	Adjustable Supports	
	Vertical V Blocks	
Clamping Elements	Side Clamps	
	Hook Clamps	
Adaptors	Riser Cylinders	
	Rack Blocks	

At the start of the fixture design process, the user is prompted for the name of the work-piece model to be used. The reusable application development classes then retrieve the geometric data of the work-piece model from the geometric modelling server to visualise and set up the model on the Java 3D canvas and scene graph respectively. The selection, transformation and interaction capabilities for the interactive fixture design application view can be carried out onto Java3D visual objects which comprises the integrated representation and placement of the Product data XML data for both work-piece and modular fixture elements.

For example, since the Java3D scene graph instantiates the equivalent of boundary representation faces of the work-piece as individual scene graph nodes, it is possible to identify these faces with the associated face tags with any relevant selection point of that face. Selection of fixture elements is supported by the concept of group node in the Java3D scene graph. Selection points are also basic to locating fixture elements onto the work-piece. These points are part of the tessellation generated from the geometric modelling server. Other associated information such as face normals, snap points and face vertices have also been interrogated and are basically used for interactive fixture design. Furthermore, the database that captures the relevant part information and dimensions of fixture elements is used to support information such as dimensions for fixture design decision making to analyse the feasibility of the use of the fixture element. Both work-piece and the elements basically are orientated with respect to each other through Java3D transformations. The middleware framework and the underlying distribution of functionality and data de-coupling approach make this possible.

On the application view, geometric and design reasoning for interactive fixture design is facilitated through the implementation of Java-based rules applying the associated Product data XML schema of the work-piece and desired fixture function and element integrated with Java3D scene graph. For instance, one general rule for placing support pins is that the triangular area formed by the three supports is the largest possible for the face selected to ensure greater stability for the work-piece. Each support pin, being an equivalent 'work-piece', has its Product data XML data with the associated snap points such as its support point co-ordinates.

The rule of deciding the triangular area can be implemented on any basis such as the maximum possible for the supporting work-piece face selected. This rule is further embedded as part of an iterative interactive structure to help user confirmation (Figure 4.19). As the rule is being used, visualization on the client end application view naturally ensures up-to-date interpretation of the entire fixture design process without further reliance on the geometric modelling server. The geometric modelling server side only needs to take care of the final transformation and assembly details.

This implementation approach also applies to the selection, analysis, and positioning of locator elements (Figure 4.20). Application view interaction without geometric modelling server access is sufficient.

Two key factors in developing an interactive fixture design system have been providing flexibility to the user in the design process and accurately reflecting constraints. The developed system provides the necessary flexibility by not constraining the user to specific locating schemes. Users are allowed to select as many

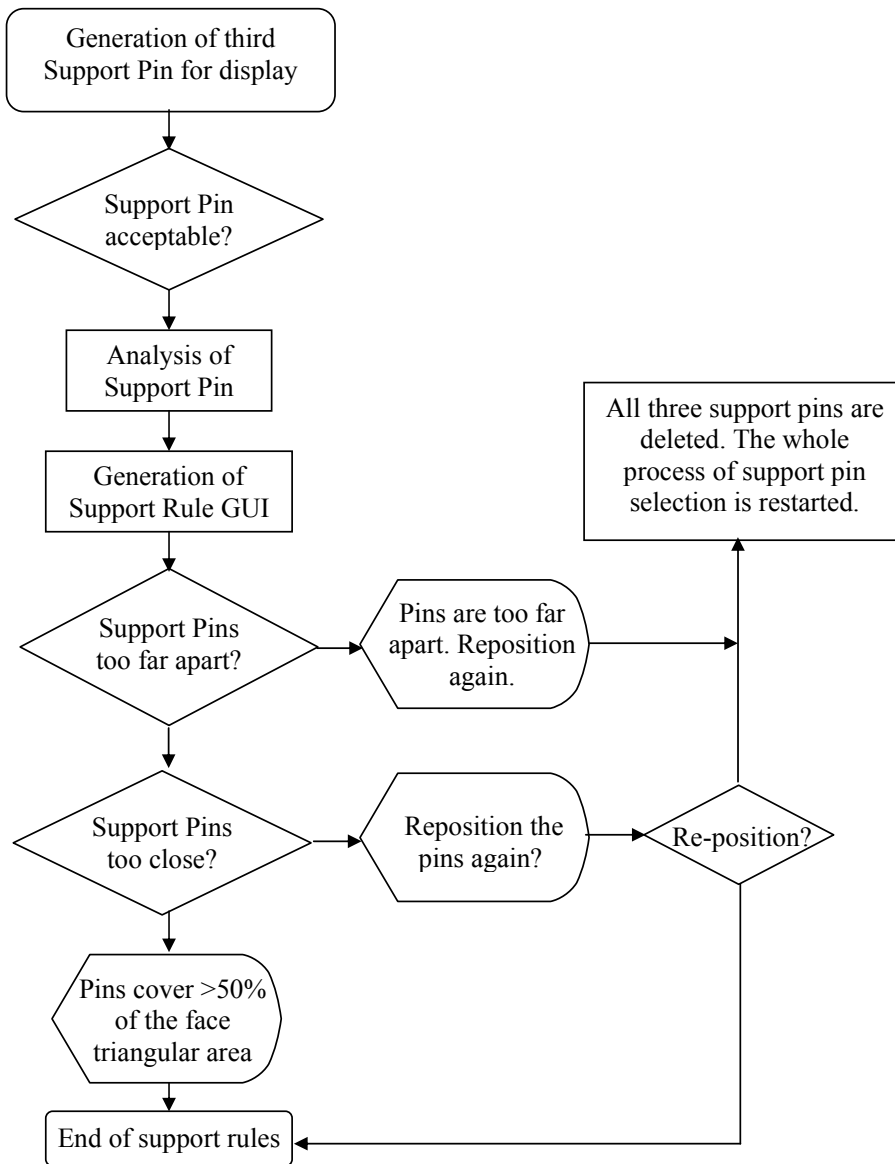


Figure 4.19: Support Rule Implementation and View Interaction

fixture elements as they want, as long as certain constraints are not violated. The constraints have been developed to be realistic and not overly constrain the design process. Occasionally, there are experienced users, in contrast with novice users, who do not wish to be constrained by say, an intelligent system that chains rules together in a fixed or rigid manner. Such users may even want built in rules to be modifiable or

customisable. Overall, the developed system is flexible and is able to guide users arrive at acceptable designs.

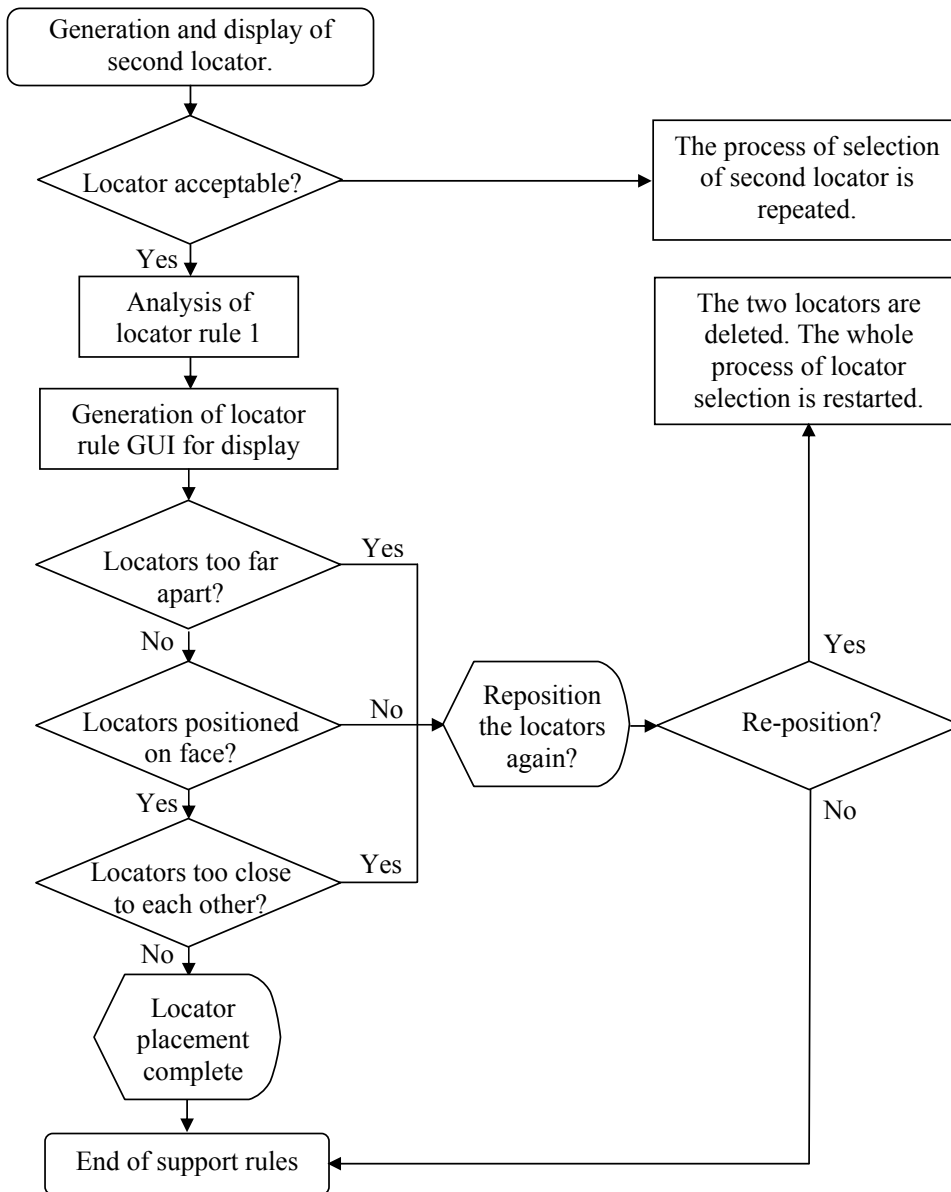


Figure 4.20: Locator Rule Implementation and View Interaction

4.5.2 Design Synchronization with Interactive Fixture Design

In a distributed collaborative design context, Figure 4.6 highlights the necessity of distributed application views during product-process interactions to be updated in a timely and consistent manner during collaboration.

The support to remotely visualize and interact with such design data requires first of all the master modeler and its geometric server to load from a repository and evaluate such design data files turning them into actual designs for collaboration. In the process of doing this, as known, a great deal of complex 3D facet dataset is generated. These datasets, also known as facet models, would be unwieldy on their own for transmission and propagation to application views. Furthermore, with design change, designs may become more complex with shape modification imposing more and more on a process application such as fixture design. This incurs the regeneration of more complicated 3D facet datasets for application views.

As a result, primary design synchronization mechanisms are needed to handle 3D facet datasets and address the nature of design change. This requires understanding and exploiting the nature of 3D facet datasets to effect more timely and consistent application view updates for product-process interaction for instance. It also requires a capability to detect design change and update product data representations prior to activating relationship management for collaborative decision-making.

For instance, should a product designer modify one side face of a workpiece as in creating an offset, a fixture designer would then need to be automatically made aware of this change in a timely and accurate manner. This awareness has to be integral as in

receiving notification about the change in the form of product-process relationships driven by the timely view update of the product model change itself to the fixture designer.

As suggested in Figure 4.15, fixture design comprises dependencies requiring accessibility considerations of face and point selections toward locating, supporting and clamping elements. The effects of design change will generally necessitate new collections of locating, supporting and clamping faces, *i.e.*, face tags that arise from design change executed on the geometric modelling server.

Underlying this update, accurately obtaining the information and data about this design change is thus vital to preparing for the application view update and relationship management for carrying out collaboration in constructing a fixture re-design response such as in re-selecting or adjusting the relevant locating element. Not doing this will mean completely but ineffectively regenerating the entire product model for application viewing. Also, important vital design change information need to be updated into the product data representation at application views for managing application relations and fixture re-design.

4.6 Discussion and Summary

This chapter has presented the development of an interactive modular fixture design system based on a middleware framework for distributed collaborative design. The architectural elements of the framework and their design purposes are based on insights into the need to appropriately distribute functionality and data, These have demonstrated the possibility of reusable application development classes toward

domain specific or discipline specific processes as application views. The classes have allowed the application developer to concentrate on the fixture design functionality, yet allowing the developed middleware framework to be seamlessly integrated with other kernel modelling systems as well.

The design of the middleware has further solved important problems faced in the development of computing environments for distributed collaborative design for integrated product-process development purposes. Through the abstraction and use of product information models via the Geometric Data XML schema, the problem of losing associated information under design changes when standard file formats are used for product exchange mechanism is resolved.

All distributed process or downstream applications thus start with having a consistent reference to geometric entities as data is obtained from a central geometric modelling server based in principle on any modelling kernel. Given the fundamental conditions facing conventional CAD systems, it is thus very much preferred for the framework to be based on a central geometric modelling server that is running and thus maintaining consistent body and face tags. This consistent reference to geometric entities is presently vital to application ability to manage design change impact without entirely repeating the entire process. This abstraction and extraction of references is the underlying means to achieving design synchronization and flexible collaboration.

This leads to the fuller context of distributed collaborative design with design synchronization which is concerned with achieving timely update and consistency of information and data across distributed environments so that interactive collaboration

can be supported. A scenario has been about how designers can interactively collaborate with fixture designers in coping with design change, *i.e.*, product-process interaction, but vital for consideration are also relevant process-product and process-process interactions where feedback and fuller product simulation take place.

In developing and incorporating an interactive fixture design via a middleware framework, it has been necessary to incorporate on the fixture design application client the necessary fixture design sequences or methodology to interact with the product model via its product data representation.

However with collaboration involving design change, it has been shown that the application client can carry out fixture design in a more adaptive manner via a mechanism that enables fixture design to be adaptive and modifiable at an appropriate stage of the sequence. Thus, [Mervyn *et al* 03b] has reported a technique of lightweight informational change management based on informational deposit and functional relationship association between applications and the product model. For example, in fixture design, the function of a locator in relationship to a specific workpiece face is captured between the fixture design application view and the product model via a relationship name 'Locating face'..

It has been assumed that the workpiece's overall size is not altered to be larger than the current base plate, and its orientation on the base plate is maintained. As such, base plate selection and workpiece set up orientation activities are not to be repeated, allowing the focus to be on the subsequent interactive choices of locating, supporting and clamping elements with selection faces and points impacted by design change.

[Mervyn *et al* 03b] indicated a need to re-import a modified work-piece in order to update the application view for consistency and accuracy prior to achieving fixture design adaptation. The effect of this re-importing is such that it is not really a live and efficient collaboration as firstly, the entire work-piece itself has to be imported and updated on the application view without the incorporation of further mechanisms for timely update and secondly, the actual (intermediate) design change itself is not captured automatically and transmitted directly via evaluating the boundary representation and topology data, even though a messaging mechanism is said to be used to transmit design change information and notification. Further, the re-importing is based on the use of the Uniform Resource Locator (URL) which means that the HTTP or Web server is the current file transport means. By the same token, modular fixture elements ranging from simple to more complex shapes and taking part in the application view also have similar considerations. Importing or re-importing work-piece designs and associated tooling geometry is in principle a form of heavy data transfer using any reasonable standard transfer protocol such as HTTP.

It is hence appropriate to investigate suitable design synchronization middleware mechanisms, fundamental to the product modeller, relating to:

1. The basic context of large amounts of complex 3D facet models for timely and consistent update of application views,
2. The basic requirement of a design change detection mechanism at the B-rep level for incremental 3D facet model and product data updates to application views in order to support collaborative decision making through relationship management and change notification.

Chapter 5

Design Synchronization Middleware Mechanisms for Effective Design Change Update

The middleware framework and application architecture development has been focused on the needs and challenges of supporting distributed collaborative design, to help realize a distributed computing environment. The approach has to be domain specific as in product design and development. It has to draw on insights into distributed (types of) functionality and data and address their issues in distributed collaborative design, such as interoperability, compatibility, distributed product and process modeling, and effective product data representations within heterogeneous fragmented value chains. The design change challenge and Internet as a distributed environment of shared resources and diverse uses require design synchronization to provide for timely, accurate and consistent application view updates, as in interactive product and fixture design. Understanding issues of CAD and geometric modeling kernel systems is vital to design change handling during product-process application interactions for effective collaborative decision-making. This chapter develops and contributes to the middleware framework design with proposed boundary representation-based design synchronization mechanisms to support design change detection and update to application views.

5.1 Design Synchronization Considerations for Application View Updates

In the present system, the middleware framework and application architecture elements are corresponded by three common classes have been developed to promote reusability: (i) remote interface class (ii) product data XML parser class and (iii) Visualization class. Based on this, extensions such as Application Relationship Management (ARM) and different applications can be added in a flexible and compatible manner.

For design synchronization, it is noted that application *views* consist of product model visualization (Figure 3.2) and the associated product data representation to support application or domain-specific tasks, *i.e.* fixture design. This visualization has been achieved by obtaining a faceted model or tessellation of the geometric model at the product modeler server. The resulting facet data has been incorporated as a product data representation using XML. This representation contains the important association of geometric face and topological information with facet data of the entire product model. In the present context, such facet data are transmitted directly without further processing to the application views to construct the Java3D scene graph and canvas, once the work-piece or part CAD file is imported or created, and evaluated on the geometric modeling server. If there is design change, the entire product data representation and faceted model would have to be used for application view update. This is not optimal and in itself, is ineffective for design change considerations.

Visualization is necessarily based on a tessellated representation of a part known as facets or polygonal meshes in 3D computer graphics terms. At the ‘user interface level’ at client computers, hardware-assisted rasterization is particularly effective at

rendering tessellated triangles [Rossignac97]. In 3D computer graphics literature, such facet data or tessellations are also known as triangle meshes. By default, a geometric modeling server, like most sources of facet data, does not optimize the triangle meshes even as it faithfully produces an accurate object rendering. This can generally result in large sets of complex 3D facet data associated with product models. As a design evolves into complex and detailed forms, increasingly complex facet data would characterize the ‘complete’ product model, *i.e.* assembled shapes with associated related tooling.

It is thus important to understand the role of computer graphics and their relationship to interactive product design and by extension, relevant synchronization issues in distributed collaborative design. As a start, this is to facilitate the choice and leverage of technique(s) or mechanism(s) for design synchronization support. In general, distributed collaborative design environments would ultimately comprise design changes and application responses.

5.1.1 Interactive Visualization in Distributed Collaborative Design

Interactive 3D computer graphics play an important role in human-computer interaction in manufacturing, architecture, petroleum, entertainment, training, engineering analysis and simulation, medicine, and science. In many of these applications, human productivity or satisfaction would be significantly enhanced by the possibility of an immediate access to remotely located 3D data sets for visual inspection or manipulation.

3D computer graphics are dominated by polygonal or facet models due to their mathematical simplicity. This results in simple effective rendering algorithms which embed well in conventional hardware leading to widely available polygon rendering accelerators. The number and complexity, measured by the number of facets, of these 3D models and data sets is growing rapidly, due to improved tools in the general context of design and model acquisition. This growth seems to be faster than the ability of graphic hardware to render them interactively. As well, anticipated increases in network bandwidth will not, by themselves, suffice to offset the explosion in combinatorial and geometric complexity of 3D models for remote access (Figure 5.1).

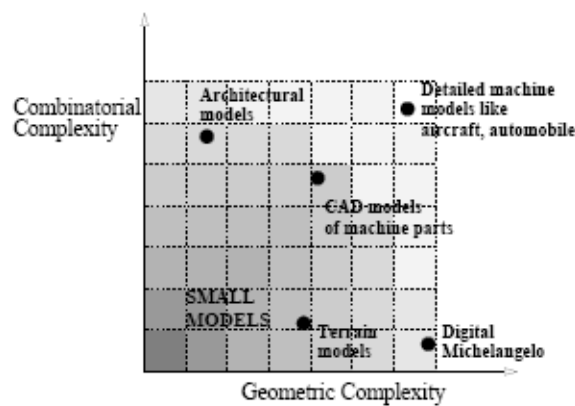


Figure 5.1: Classification of 3D Models – Geometric Complexity vs Combinatorial Complexity [Shikhare 01]

This observation directly applies to design and manufacturing as products have grown in wide-ranging variety and complexity. Given the distributed collaborative design context, there is always a need for product models and parts assemblies to be remotely viewed and operated on interactively. With this visualization perspective, network transfer is bound to be data intensive compared to say, informational application relationships handled by the ARM technique storing functional relationships *e.g.* “Locating Face” between product model and application views like fixture design.

Also, it has been highlighted before that it is better to simply avoid the need to pass pixel or image data across the network. Pixel or image data is large quantity of non-object data that required many updates (each update involves the entire object scene) across the network. Instead, transferring object-based 3D graphics or faceted models to the visualization functionality of the application client for interaction, manipulation and application processing would be far more effective.

Hence, graphics handling or simplification techniques as enabling technologies are relevant to distributed collaborative design. A brief critique relevant to the particular needs of distributed collaborative design follows. For reference, a good survey of such polygon simplification techniques is [Cignoni *et al* 98].

5.1.2 Graphics Simplification Techniques

Briefly, these methods can simplify polygonal geometry of small, distant, or otherwise unimportant redundant parts of objects, seeking to reduce the rendering cost without a significant loss in visual interpretation, as in a flight simulation of dogfights. Alternatively, these methods can reduce model complexity without introducing geometric error such as in volumetric information stemming from medical imaging useful for surgical simulation. In the case of complex engineering analysis and simulation problems, a model is required to go through subdivision or partitioning, and simplification is then employed to remove unnecessary geometry. If the problem is to improve runtime visualization performance by simplifying the polygonal scene, the most common polygonal simplification technique is to generate levels of detail (LODs) of the objects in a scene [Lindstrom 96]. By representing distant objects with a lower LOD and nearby objects with a higher LOD, applications

from video games to CAD visualization packages can accelerate rendering and increase interactivity. In the latter, this would be evident in factory simulation rather than product design due to the spatial scale and multitude of objects involve in factory design and planning. Losses in geometric accuracy and details can be tolerated.

In distributed collaborative design, design access, changes and updates need to occur across distributed environments. When synchronization is considered, simplification techniques requiring time consuming preprocessing effort to generate multiple LODs would not be suitable. Such LODs will mean multiple updates are required for distributed design, even though LODs can be progressively transmitted or streamed [Hoppe 96]. Worst, LODs severely compromise the geometric and visual fidelity required in product design and would not even be advisable in co-design involving distributed teams members.

Nonetheless LODs have been a key influence behind the design and specification of the Virtual Reality Markup Language (VRML) standard and other programmatic scene graph methods for visualization. Their original context has been much more related to visual simulation and multimedia uses. It is thus inappropriate to distributed collaborative design context.

In addition, simplification techniques that drastically allow for topology modification, compromise or loss [El-Sana *et al* 97] [Schroeder 97] are also inappropriate in distributed collaborative design. They also create inaccuracy and inconsistency in the original CAD topology and geometry or boundary representation. The resulting model for product design would then be grossly misinterpreted when design features are

'lost' during communication. Unlike flight or factory simulation, collaborative design requires a more static or stable viewpoint, as opposed to dynamic visual simulation of large spaces with many objects needing pre-computation for real time scene updates.

In addition, distributed collaborative design requires an integrated framework supporting online dynamic product-process collaboration characterized by design changes. Collaborations are also virtual networks carried on the Internet. Therefore graphics simplification techniques that are transmission or bandwidth friendly without losing accuracy must be considered. Ultimately, this is for application view accuracy and consistency.

It is also noted that during distributed collaborative design, design changes occur and thus, the geometric model is actually changing and evolving. For this reason, previously mentioned simplification techniques are also inappropriate as the scene model is actually changed and the entire polygonal mesh needs to be re-processed or re-simplified. This view is similar in understanding of the costly challenge of history maintenance of managing change, *i.e.* rolling back and forth, references to geometric and topological entities in the B-rep model of conventional CAD systems.

An algorithm is thus required that generally takes in original highly detailed and complex models and reduces their sizes to a bandwidth-acceptable level of complexity without compromising visual and topology fidelity. Without this considering this as part of design synchronization, distributed collaborative design would be impractical.

5.1.3 Graphics Compression Algorithms

Much of the work done in the area of geometry compression is based on innovative topological encoding schemes of the connectivity between facet vertices or nodes in the meshes. These encodings set out to minimize the repeated references to nodes, the main source of complexity, thereby achieving a compact description of topology. An interesting but realistic observation made in meshes representing manifolds is that, on an average, the number of triangles is twice the number of vertices and each vertex is referenced in 5 to 7 triangles. Hence, a lot of research has concentrated on aggressive attack on the problem of encoding of topological relationship between vertices.

Early examples of compact encoding of a mesh were seen in triangle rendering engines such as OpenGL, in the form of triangle-strips and triangle-fans. A lot of research has been carried out in generating maximal triangle-strip decomposition of given meshes, minimizing the repetitions in the references to vertices.

Concerning more contemporary means of compression, the original work started by [Deering95] described a more efficient “generalized triangle mesh” representation for further reducing the redundant referencing of vertex data. A set of four operators is defined to interpret a stream of vertex indices referring to the list of points. This leads to an efficient encoding. For the present purpose, Rossignac’s work on the Edgebreaker algorithm achieves even greater compression by compact representation of topological relationship between vertices of a mesh [Rossignac99].

Hence, the suitable choice of model or geometric compression is based on the following factors: 1.) that it is basically lossless compression - topology preserving and geometrically accurate, and 2.) that it is capable of high compression ratios relevant to the bandwidth and transmission constraints of the Internet as a shared resource and yet, expedient to the need for timely updates to application views.

Model compression applies also to the context of assembly models or assemblies of parts. For example, in the case of fixture planning, both workpiece and various fixture tools can be subject to model compression. This suggests that a multi-prong approach will be needed in a more general context of collaborative product development.

Hence geometry data transmission speed is critical to expediting interactivity in a distributed collaborative design environment. Moreover, a compression scheme also reduces storage costs *i.e.* the tessellations as repositories can be more readily available in highly compressed forms. Loading a work-piece and say, an entire fixture design assembly onto application views and product modeler servers can be more readily and simultaneously carried out.

What is subsequently important is when design changes actively take place, the role of integrated model compression for application view update becomes crucial. This aspect has not been reported elsewhere in the distributed collaborative design context. In conclusion, in terms of design synchronization relating to timely updates for application views, size reduction to store and/or transmit such 3D models or datasets is thus expedient if not crucial.

This is where model or geometric compression is suited for – integrated compression on the geometric modeling server side, followed by transmission and decompression into a proposed enhanced augmented product data XML representation. Thus, a key early consideration for the middleware framework and design synchronization is the fast compression and decompression of 3D faceted models involving innovative topological encoding schemes and bit efficient facet formats.

In the distributed collaborative design context, appropriate distributed functionality and data with associated mechanisms for manageable network loads are important considerations as noted in [Bidarra *et al* 01] [Li and Qiu 06]. It is thus proposed that geometric or model compression is important and useful technique as a design synchronization mechanism within the middleware framework.

5.2 Leveraging Model Compression for Design Synchronization

5.2.1 Model Compression Algorithm

Model compression via the Edgebreaker algorithm [Rossignac 99] thus works to reduce the storage size needed to record triangle meshes. A triangle mesh may be represented by its vertex data and by its connectivity. Vertex data comprises coordinates of all the vertices and optionally the coordinates of the associated normal vectors and textures.

In its simplest form, connectivity captures the incidence relation between the triangles of the mesh and their bounding vertices. It may be represented by a triangle-vertex incidence table, which associates with each triangle the references to its three bounding vertices. Connectivity compression is usually achieved by reducing repeated

references to vertices that are shared by many polygons/triangles. A brief description of the Edgebreaker algorithm follows.

For all meshes homeomorphic to a sphere, and in fact for most meshes in practice, it has been observed that the number of triangles is roughly twice the number of vertices. This means that typically a large part of the representation of a model is the definition of the connectivity. Also, when pointers or integer indices are used as vertex-references and when floating point coordinates are used to encode vertex locations, connectivity data also consumes twice more storage than vertex coordinates leading to the use of bit-efficient quantization schemes as well. Schemes that minimize repeated references to vertices would additionally result in compression. These include using half-edge data structures to carry out history-based and opcode-based tree-traversal to encode mesh connectivity [Rossignac99].

Fundamentally, the Edgebreaker algorithm works in two stages: an initialization process, followed by the compression process. In the initialization process, the Edgebreaker is further broken up into more steps: a marching process, followed by marking of the bounding edges and vertices of the mesh, and a stack initialization. Initialization formats the geometry data and marches throughout the bounding edges and vertices to establish the connectivity between all the labeled triangular meshes. The compression is a recursive procedure that traverses the mesh along a spiraling triangle-spanning-tree and encodes the vertices and connectivity to generate compressed model.

In general, Edgebreaker adopts a simple representation of the compressed data as a set of {**C**, **L**, **E**, **R**, **S**} encodings that indicate 5 exclusive states of each triangle's relation to the mesh boundary (Figure 5.2).

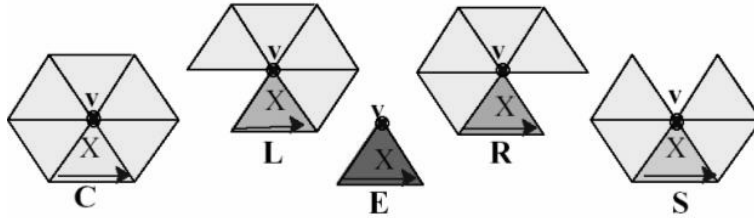


Figure 5.2: CLERS Illustration [Rossignac99]

The different triangles represent different cases in CLERS. The triangle X is formed by the gate edge **g** and vertex **v**. The location of **v** with respect to the boundary **B** determines the operation type: **C** (**v** is not on **B**), **L** (**v** immediately precedes **g**), **R** (**v** immediately follows **g**), **E** (**v** precedes and follows **g**), and **S** (**v** is elsewhere on **B**).

During compression, Edgebreaker will determine the gate **g** which will be made the starting vertex of the compression sequence. This gate, being on the boundary, will be loaded onto a stack, to initiate the compression. The basic idea is that Edgebreaker starts with the gate and proceeds or traverses to march round the edge of each triangle. Edgebreaker will proceed to determine whether the triangle represents a **C**, **L**, **E**, **R** or **S** case based on where the vertex is in relation to the boundary (Figure 5.2).

The determined case is stored as an alphabet or integer code in a compression history List, **H**, after which, Edgebreaker will proceed to the next triangle. It will determine again the relevant case, and then loading this case into the **H** list. This process goes on, until the Edgebreaker has fully stored every single triangle in the **H** list. There will

be points along the compression process where there will be a loading of new stacks of gates into a vertex list **P** to allow for the compression of new regions of facets that the previous series of compression was unable to reach. This compression process removes all triangles of the mesh and always terminates, because the preconditions for the L, R, C, S, and E operations are mutually exclusive and cover all possible cases, and because these operations all decrement the triangle count in the mesh (Figure 5.3). These lists are subsequently further encoded using binary code schemes and are then handled by the decompression process [Rossignac 99].

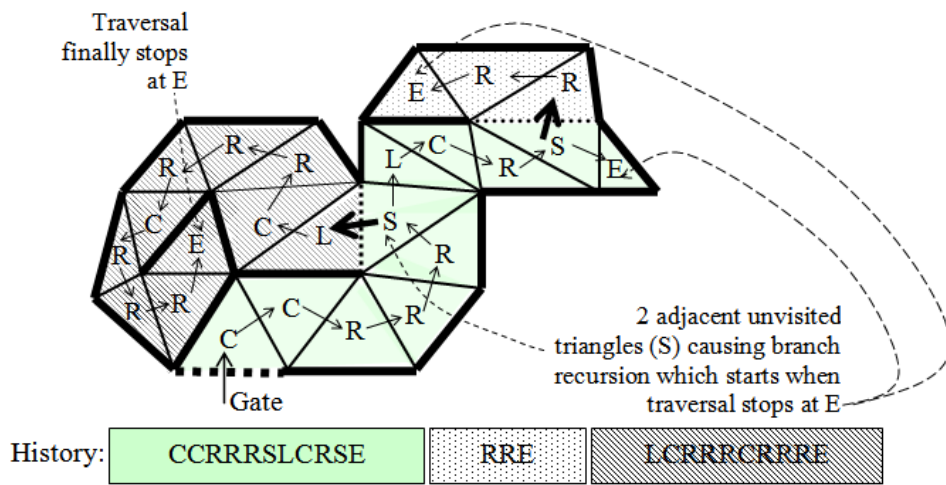


Figure 5.3: Model Compression Traversal

Subsequently, model decompression is roughly a reverse process of the model compression. It includes decompression process and post process. The decompression process reconstructs the mesh from the input streams. The post process also recovers any holes (generated from internal bounding loops of the boundary representation topology) and duly converts the data formats. Figure 5.4 illustrates the compression and decompression modules and internal operations.

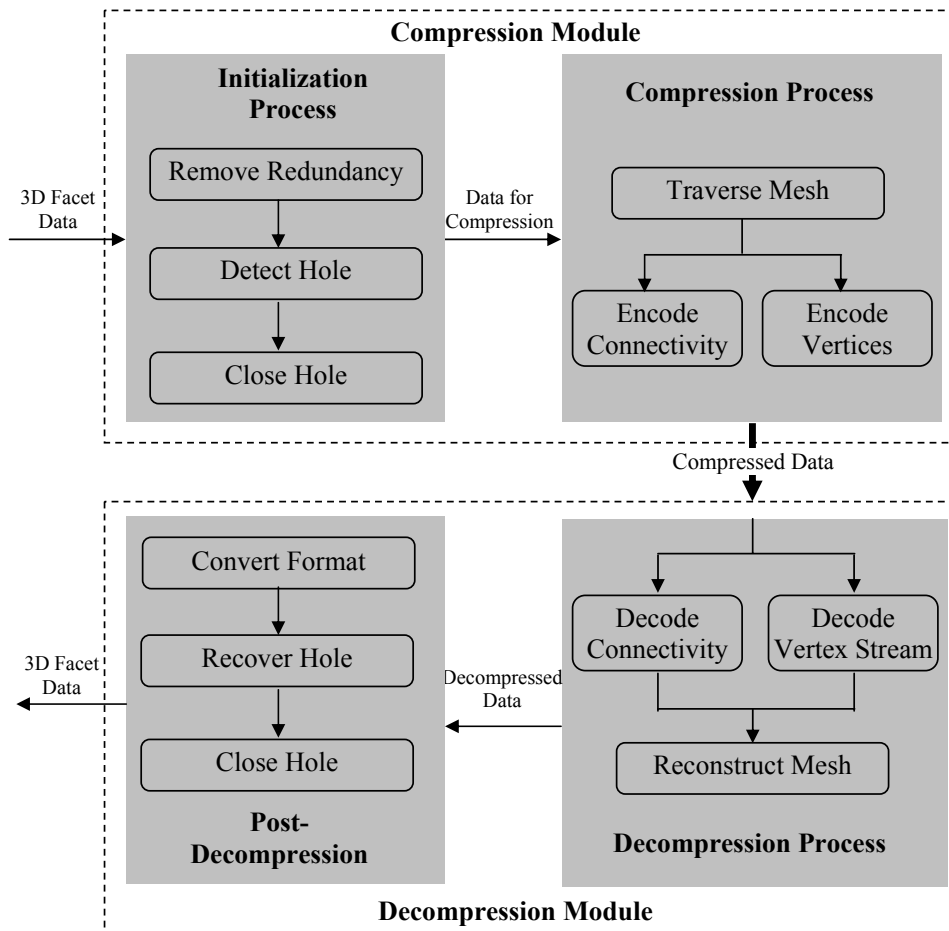


Figure 5.4: Model Compression and Decompression Procedures

5.2.2 Product Modelling Architecture with Integrated Model Compression

With the focus on the incorporation of the model compression technique using the Edgebreaker algorithm, the sequence of events is modified such that when an import work-piece or modeling operation is carried out, the geometric modeling kernel-tessellated mesh of the entire model is converted, re-formatted and integrated into the model compression algorithm. The mesh data required for Edgebreaker are the number of vertices, the coordinates of the vertices, the number of triangles and the indices of the vertices that belong to each triangle. The resulting sequence of events

for arriving at the augmented Product Data XML schema is as shown (Figure 5.5).

The augmented Product Data representation is discussed shortly.

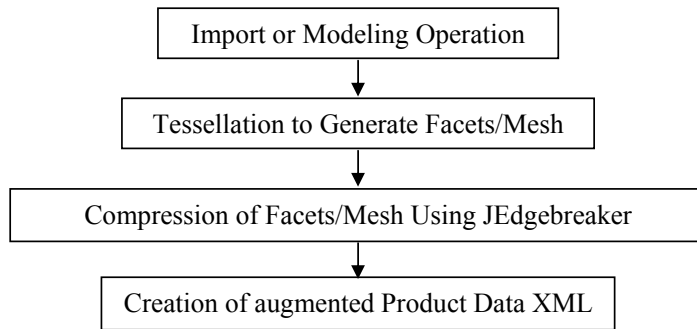


Figure 5.5: Basic integration and sequence of creating the augmented Product Data schema

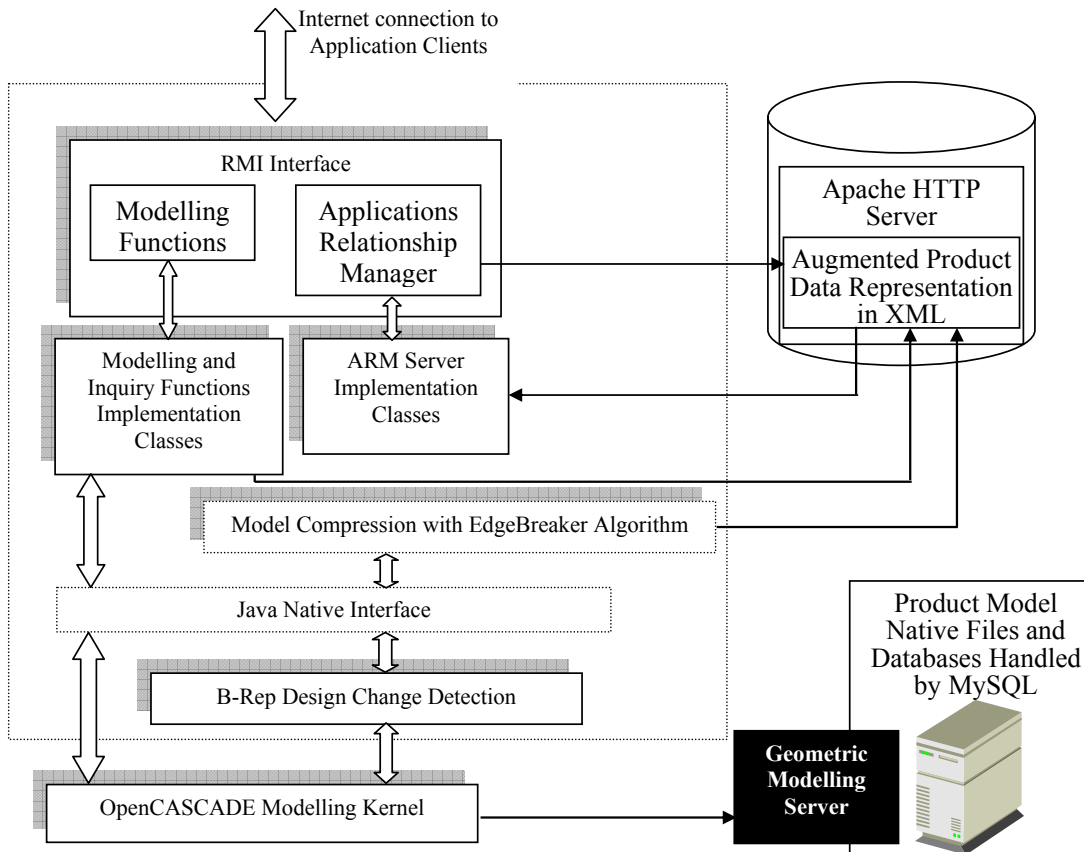


Figure 5.6: Product Modeler Architecture with Model Compression and Design Change Detection

It is thus proposed that the model compression technique should be integrated into the middleware framework as a design synchronization mechanism for the product modeler to help achieve distributed collaborative design. With the benefit of the reusable application and interface classes, the resultant product modeling server architecture with integration of the model compression technique is indicated in Figure 5.6. In this particular context, a Java implementation of the Edgebreaker algorithm, known as JEdgebreaker is used to improve the middleware framework, and as a result, an overview of the product modeler system architecture is illustrated.

5.2.3 Augmented Product Data Representation

Based on the above understanding of model compression, a modified augmented Product Data XML Schema has been developed to handle compressed geometry data for complete parts, considering that these parts, *i.e.*, workpiece and tooling, can be retrieved from a repository.

The Edgebreaker algorithm was mainly developed for visualization of complete 3D models independent of the origin and role of those models. In the context of distributed collaborative design, the compressed data format and encoding need to be related to be incorporated into the product data representation thus far for application views and their updates.

To incorporate model compression as a design synchronization mechanism, the product modeler supports an enhanced format introduced as the augmented Product Data XML representation (Figure 5.7). The revised form of the XML schema is described as follows.

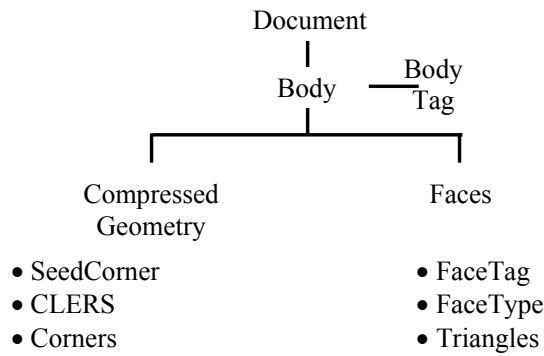


Figure 5.7: Augmented Product Data schema incorporating compressed geometry

Compared with Figure 4.21, the essential difference in the product data representation is the displacement of the <FACET> sub-branch tag with a re-organization and augmentation of the compression encoding sub-tags under a new ‘branch’ tag. Basically, the <SEEDCORNER> sub-tag defines the starting corner of the compression sequence. The <CLERS> sub-tag contains a record of the compression history list where each facet is being reached by the compression algorithm. The <CORNERS> sub-tag contains the coordinates of the vertices (corners) in the original model. The <FACE> branch tag now only contains model information at the face level i.e. face tags and face types like in Figure 4:12. The <TRIANGLES> tag contains the indices of the triangles that belong to a face – basically providing access into the facet data themselves.

In this way, the modelling information is not lost due to the compression as associations are still maintained and a compact representation is still obtained. [Bok *et al* 04]. The <COMPRESSEDGEOMETRY> tag therefore refers to the compression encoding of the tessellated model which has become a compressed data. An illustration in Figure 5.8 shows how the augmented Product Data representation is instantiated.

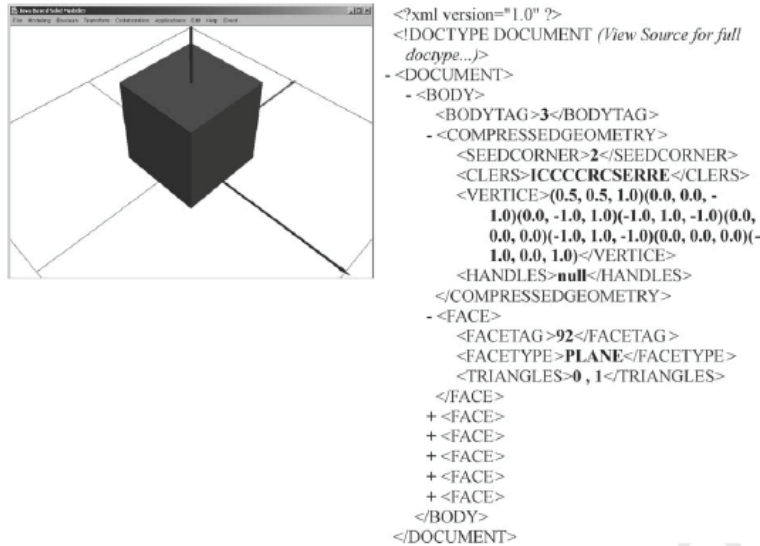


Figure 5.8: Illustration of Augmented Product Data schema

The visualization class within the reusable application classes layer thus retrieves information from the data structures and decompresses the encoded information from model compression, *i.e.* CLERS, vertices and handles data into facet data. The facet data is then to be sent to Java3D classes for rendering into the Java3D canvas, which will also be set up by this class. It is also noted that the augmented Product data XML schema is in itself already flexible whether for representing the entire workpiece or just an individual face. The next section deals with leveraging model compression in the context of design change with this flexibility in mind. Figure 5.8 also illustrates the presence of the <COMPRESSED GEOMETRY> record.

As facet models are characteristic of engineering and product models, they should be kept in a repository associated with the product models or native CAD files. Essentially in compressed data formats, such facet models via the augmented Product Data representation can facilitate a new or re-started collaborative session as native file data are drawn from the repository and instantiated into the geometric modeling

server. The augmented Product Data can be concurrently transferred to application views along side with the instantiation of the product modeler server in a networked distributed environment.

5.3 Experimental Results of Integrated Model Compression

The effectiveness of the Edgebreaker algorithm has been initially verified by simply comparing XML file sizes based on the original product data representation (with the uncompressed <FACETS> sub-tag) and one that stores the compressed data within with compressed geometry format for convenience. Based on some basic primitive models, and at coarse resolution, chuck and flange examples (Figures 5.9 and 5.10), the results are presented in Table 5.1.

Table 5.1: Size reduction tests with model compression

Model Type	File Size		Compression Ratio
	Before Compression	After Compression	
Cube	5KB	1KB	5
Prism	7KB	2KB	3.5
Sphere	178KB	31KB	5.74
Torus	379KB	66KB	5.74
Chuck	492KB	79KB	6.23
Flange	137KB	39KB	3.5

The experimental results show a significant compression of the data required, proving the effectiveness of using the Edgebreaker algorithm for model compression in reducing data sizes. Data compactness is a vital requirement due to the shared bandwidth nature of the Internet and thus 3D facet models or triangle meshes should be made as compact as possible.



Figure 5.9: A Chuck Workpiece

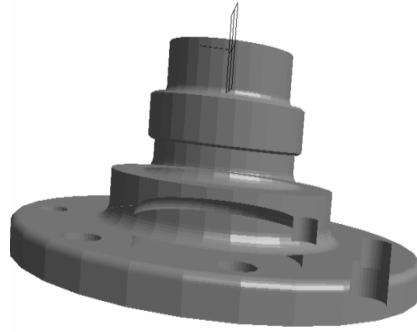


Figure 5.10: A Flange-like Workpiece

Initial timing tests were also conducted to compare visualization on the application view were also taken to verify the effects of compression on prismatic and non-prismatic models of different facet resolutions or complexity (Table 5.2).

Table 5.2: Timing tests for visualization

Model Type	Visualization Time		% Difference
	Without JEdgebreaker (secs)	JEdgebreaker (secs)	
Cube	1.74	1.74	0.0
Prism	1.49	1.49	0.0
Sphere	6.73	3.53	47.5
Torus	13.07	4.00	70.0

The results demonstrate that using compression (and decompression) for application view update is more pronounced with greater facet complexity. However, as the Internet is an increasingly heavily shared (common) resource and computers themselves nowadays are also used for concurrently for many formal and informal purposes, a fully adequate predictable performance metric is not really possible.

Hence it may be expected that commercial private network services with feasible 'Quality of Service' (QOS) management policies will come to ensure the viability of

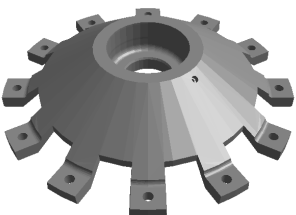
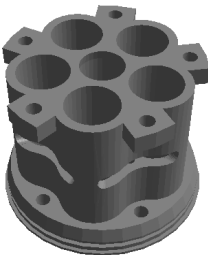

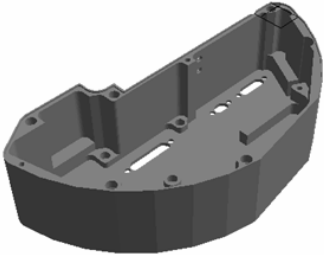
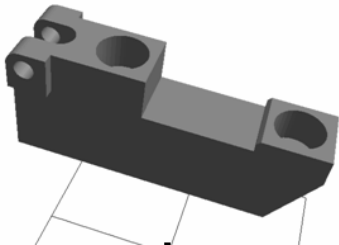
	<u>Before Compression</u>	<u>After Compression</u>
Mesh Size:	986 kB	142 kB
	<u>Before Compression</u>	<u>After Compression</u>
Mesh Size:	643 kB	93 kB
	<u>Before Compression</u>	<u>After Compression</u>
Mesh Size:	4.76 MB	678 kB
	<u>Before Compression</u>	<u>After Compression</u>
Mesh Size:	789 kB	114 kB
	<u>Before Compression</u>	<u>After Compression</u>
Mesh Size:	120 kB	17 kB

Figure 5.11: Additional Results of Integrated Model Compression

distributed and collaborative design. Additionally, the following are demonstrative engineering examples of integrated compression of differing complexities albeit with a coarse facet resolution (Figure 5.11). The comparison is based on mesh data sizes directly obtained from JEdgebreaker.

Thus far, the discussion has been about the feasibility of model compression for distributed collaborative design in the context of invoking complete product models from a repository for application view updates. For collaboration involving design change *i.e.* shape modification, the role of model compression operating in tandem with design changes themselves is crucial to the ability for application views to be updated incrementally or progressively, *i.e.*, only the changes should be dealt with rather than altogether.

The next section deals with this to provide for a distributed collaborative design environment that has the integral role of model compression and design change handling to support collaborative decision-making, as in design change causing fixture re-design. In this manner, it can be said that design synchronization middleware mechanisms are capable of supporting design change across distributed environments. The product modeler server architecture in Figure 5.5 is referred to.

5.4 Design Synchronization for Design Change

The DET and Distributed Collaborative Design issues highlighted in Chapter 2 compel the need for infrastructures and mechanisms to manage distributed functionality and data for collaboration. Infrastructures require application architecture and middleware framework to address heterogeneous and diverse tools

and computing environments that basically impede collaborative product development and manufacturing amongst users. However, with design change involving shape modification, middleware mechanisms are required to further enable the product modeler server to effectively support application view updates (Figure 5.5). Updating depends on effective update and visualization of facet models accompanying the augmented Product Data representation.

The following section deals with the verification of the model compression technique to support design change. Following that, the fundamental issue of design change handling at the shape modification level needs to be addressed to support compression. This requires B-rep evaluation mechanisms operating at the core of geometric modeling kernels. It should be regarded as middleware since product models commonly or universally relate to geometry entities such as faces and vertices, and their topology in boundary representations.

5.5 Local Face Model Compression for Design Change Synchronization

The present purpose in the context of application view update is to ensure that the faceted model of the design change can be compressed. This section here verifies that model compression can be applied at the face level as opposed to the entire product model, as presented earlier. This is experimentally done by manually selecting and retrieving a face to obtain its faceted model as an input to test the JEdgebreaker compression module. Figures 5.12 and 5.13 demonstrate this interactively with the result of compression at the product modeling server.

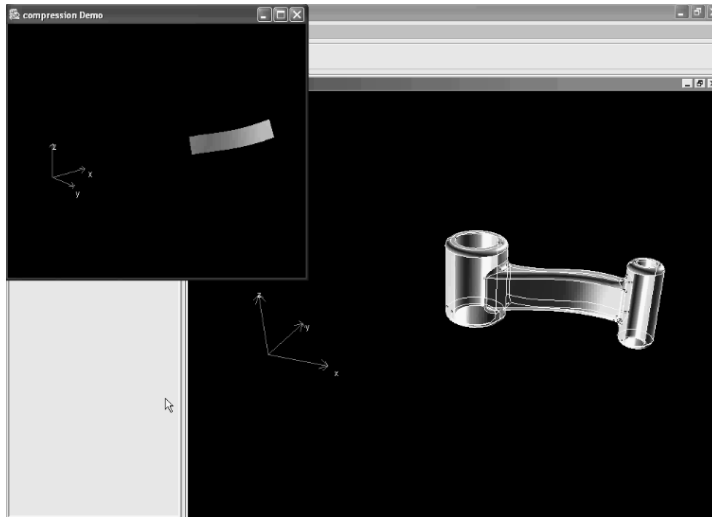


Figure 5.12: An interactive demonstration of face selection for compression

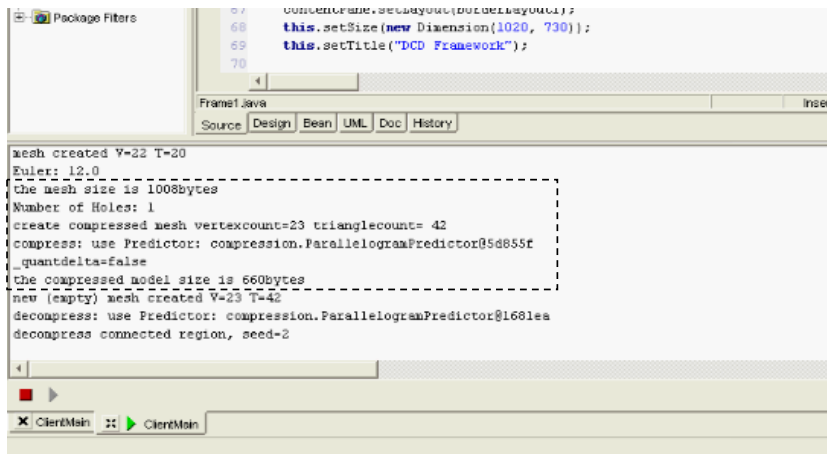


Figure 5.13: Corresponding face compression results

In addition, Figures 5.14 and 5.15 illustrate the ability to compress face meshes after a user-invoked design change operation, *i.e.* fillet command. This is to be expected once the B-rep model is successfully evaluated. Compression at the face level is thus possible as it has been verified that the JEdgebreaker can process the triangle mesh of a face in the same manner as the entire facet model of the whole workpiece. Given the

augmented product data representation, the DTD has provided the flexibility for each compressed mesh to be associated with each face as highlighted before.

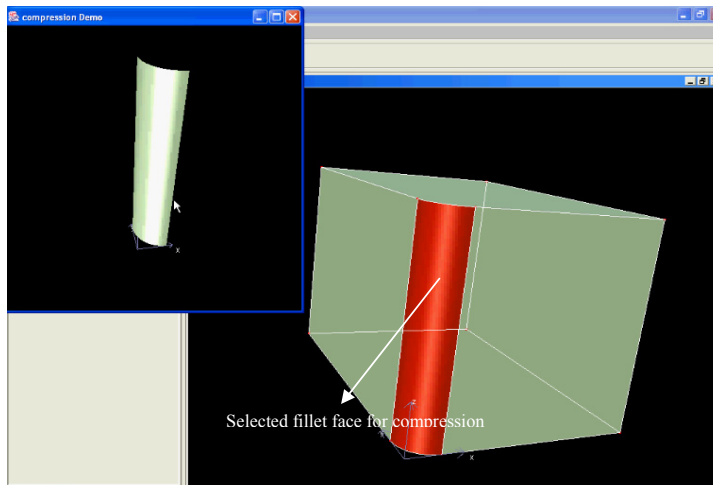


Figure 5.14: Interactive fillet modeling operation with compression of selected generated face

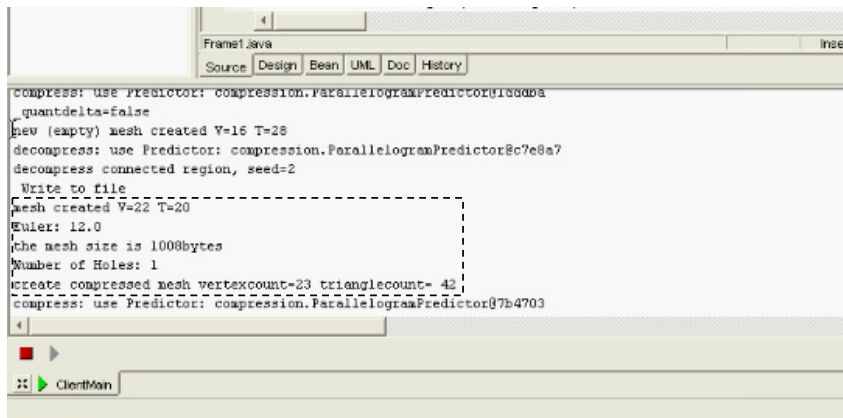


Figure 5.15: Compression of face mesh corresponding to fillet operation

5.6 Design Change Detection within Shape Modification

Design change can involve extensive shape changes with one or more faces modified and replaced, newly generated or removed in the boundary representation of the product model. By detecting these faces and understanding what takes place in the boundary representation, and knowing that model compression at the face level can take place, the groundwork is laid for an integrated approach to accurate and

consistent design change handling.

To be more precise, design change can be described as deltas or variable data in the topology and geometry of a boundary representation when shape modification occurs to create new boundary representations. In the context of face shape entities, this delta is classified and described as follows:

1. New face shapes generated with new reference tags, new surfaces and facet data,
2. Face shapes modified/replaced with new reference tags, modified surfaces and facet data,
3. Face shapes unmodified and mapped with new reference tags but the same surface and facet data, and
4. Face shapes removed with facet data no longer necessary to the application views.

Specifically, when a boundary representation is re-evaluated, reference tags are updated or adjusted by the geometric modeling kernel. They cannot be assumed to be persistent even as static CAD files are not dealt with i.e. imported into the kernel for tags to be re-emitted. Due to the nature of XML schemas, the augmented product data representation can be leveraged to capture the abovementioned information related to this delta.

Without incorporating an integrated approach to design change handling, *i.e.* detection and update of this delta information, conventional interactive design modeling operations based on kernel library routines would just execute and provide a visualization update of the entire model. It would just be equivalent to an exercise in

creating an interactive user interface with commands to invoke modeling operations not unlike in conventional CAD systems. However, this only means that there is no change detection explicitly taking place during runtime to support distributed collaborative design.

In other words, design change detection has to take place explicitly during shape modification within the kernel to enable design synchronization with application views. The boundary representation has to be processed together with the important re-emission and evaluation of the topology and geometry of shape entities.

A change detection algorithm should therefore be incorporated into a geometric modeling kernel at runtime in order to capture and process old and new B-rep models to discover the delta information. In particular, surfaces that result from new generated, or modified and replaced faces are important to derive faceted models for more effective application view updates. Faces that have been removed or deleted cannot result in a faceted model for application view update. Also, without capturing the delta information, it would be impossible to carry out design synchronization at the application relations management level, as discussed in the next chapter.

5.7 Boundary Representation Model Changes

Usually, a product model is defined and described with geometric shapes or entities that are organized by an internal hierarchical topological data structure without which basically, solid models and modeling operations cannot be digitally realized. This tree-like data structure is typically organized as from root to leaves and is core to the boundary representation of the product model and the connectivity of its shapes based

on topological common boundaries. It is vital to the product model in terms of defining and maintaining the integrity of the underlying solid model geometry and its properties for solid model reasoning in various applications.

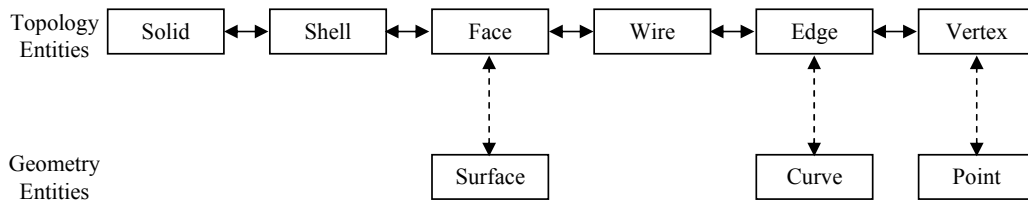


Figure 5.16: Boundary Representation Graph Model

A root refers to the solid model or body itself which is then recursively represented by the following topological entities or types in the data structure: Shell, Face, Wire, Edge and Vertex. Each entity refers to its appropriate geometries (Figure 5.16). A vertex entity would essentially be a point. An edge entity would refer to a geometric curve, line or a connected line segment. A wire shape entity essentially represents the boundary or closed set of edges of a face. Ultimately, the shell entity is topologically the container of the recursive subsidiaries of Face, Wire, Edge and Vertex entities that form the solid model.

The product model's B-rep is dependent on its run-time state during the kernel's execution. The shape entities are maintained in the form of runtime internal maps of respective types. These maps comprise indices that address essentially the kernel's internal shape entities. Indices change dynamically due to the fact that B-rep operations are applied interactively resulting in shape entities of different states that need to be updated into the maps. These updates are internal to the kernel and hence the maps cannot be explicitly manipulated. These maps are part of the runtime B-rep evaluation process and are not contained inside a native geometry or CAD file.

However, as long as geometric kernel modeler supporting the product modeler server is running, the ordering of shape entities during modeling would be consistent with the sequence number of shape entities. They are different from shape tags as they can change due to B-rep state operations. Shape tags can thus be assigned as an integer identifier for each shape entity type in an application development.

This behavior contributes to the persistency problem associated with geometric kernels or modelers as they do not inherently support persistency of shape tags or ids associated with each type of shape entity. Such tags or ids are inherently important to product modeling as designers and engineers deal with product shapes and features; even to the extent that for instance, feature representations are viable once such tags or ids are represented and persistently used in design modeling operations.

The run-time geometries underlying these entities are maintained differently through the respective indexed maps. The index of each shape entity is essentially the tag for that shape entity. Creating or re-making a shell is hence a vital step to creating and evaluating a new boundary representation. The above description is fundamentally the case in today's mature geometric modeling technologies.

In general, it is known that a new boundary representation model when evaluated during the running of a geometric modeling kernel can be used to write out its contents in the form of a series of Shell, Face, Wire, Edge and Vertex entity maps.

With shape modification, there are two types:

1. Geometric modification: the initial solid is modified without changing its topology, *e.g.* modifying a hole diameter without any interaction with other surfaces
2. Topology modification: the topology of the solid model is modified, *e.g.* changing a chamfer to a fillet.

During a shape modification, the kernel performs a B-rep evaluation which involves an updating of the shape entities from the old B-rep model to the new one. The updating comprises an internal graph of topological operations on shape entities. The operations are:

- ‘change’ – a shape entity is modified from an old model to a new model in terms of its local attributes:- position, area (face), length (edge) and centroid
- ‘merge’ – two or more shape entities of the same topology type are combined into one entity
- ‘split’ – two or more shape entities are created from one entity
- ‘add’ – a shape entity is created
- ‘delete’ – a removal of the entity
- ‘mapping’ – a shape from the old model is not changed in the new model, only its internal index or sequence number is updated

The first three operations are normally subsumed under a ‘modify’ operation.

Figure 5.17 illustrates these operations with the following convention: ‘fa’ as face, ‘ed’ as edge and ‘ve’ as vertex, the integer numbers representing sequence number of the shape entity type in each internal index map, and the shades mean the shape entity is newly added.

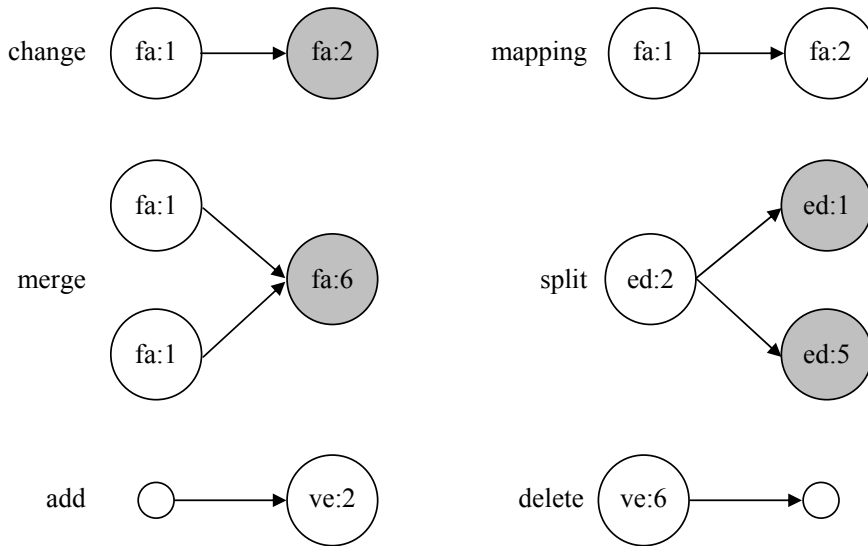


Figure 5.17: Illustration of Types of Topological Shape Changes

In general, a shape entity in a new B-rep model can be treated as modified, added, or mapped, and one in the old B-rep model can be modified, removed or mapped. Thus with design change or shape modification, the need is to decide the add(), remove(), modified()/replaced() and even mapped() collections of indices to entities (which can be identified by sequence numbers or tags from the kernel) during a runtime

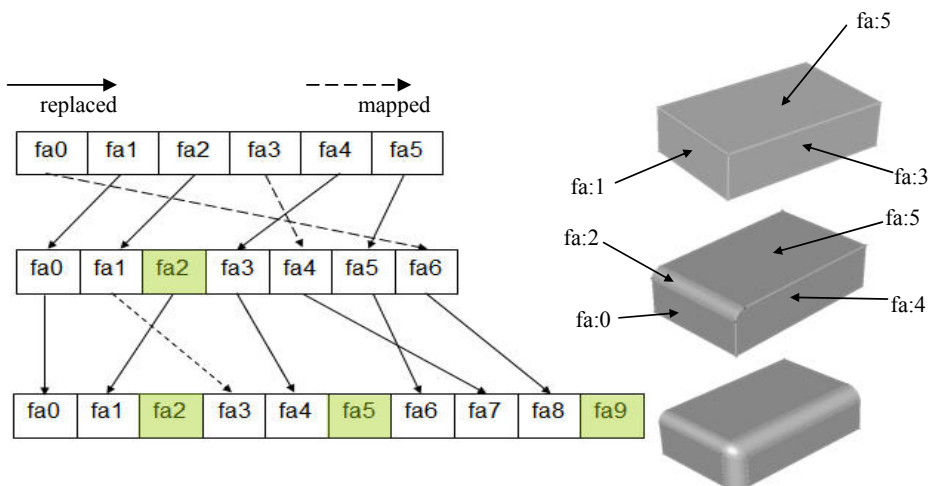


Figure 5.18: Illustration of B-rep face shape entity state changes

execution of a modeling operation, as in making a fillet.

Figure 5.18 illustrates the B-rep face entity state changes of add() i.e. the shaded entities, modify()/replaced() and map() i.e. re-assignment of tag references for fillet modeling operations. Figure 5.19 correspondingly shows the detailed entity state changes in the first B-rep fillet operation. The modified() shape state actually means replacement of shapes (e.g. fa:1->fa:0) due to shape geometry changes in local design change, such as in the ‘curtailing’ of the side faces adjacent to the fillet, as well as the top and bottom faces of the block. In other words, there is no such state as a removed or deleted for an old face so to speak, rather that the face has been geometrically modified and replaced back with a face tag update. It is noted that vertices can only have add() and remove() states.

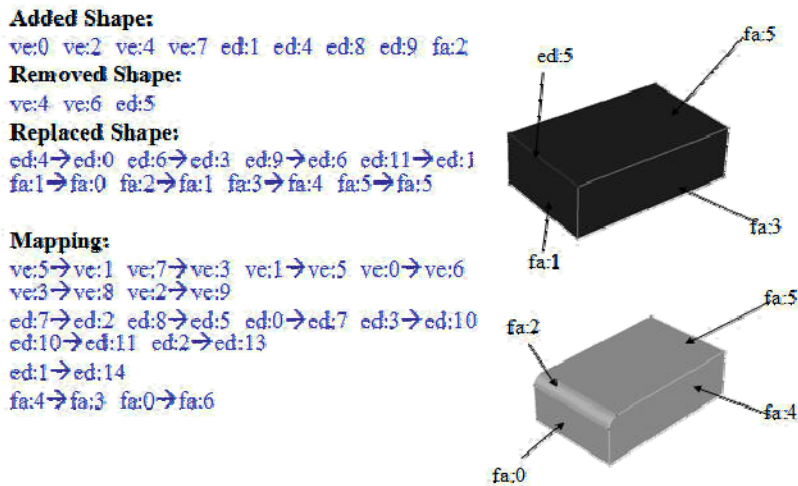


Figure 5.19: Illustration of B-rep shape entity operations inside design change

5.8 Boundary Representation-Based Design Change Detection

It has been indicated that design change detection would need to be carried out inside a shape modification process ‘intervening’ into the B-rep model’s shape entities. As

well, B-rep models are evaluated during a kernel's runtime. An integrated approach is thus needed to support shape modification with design change detection. Figure 5.20 shows the sequence of steps for this integrated approach to be realized.

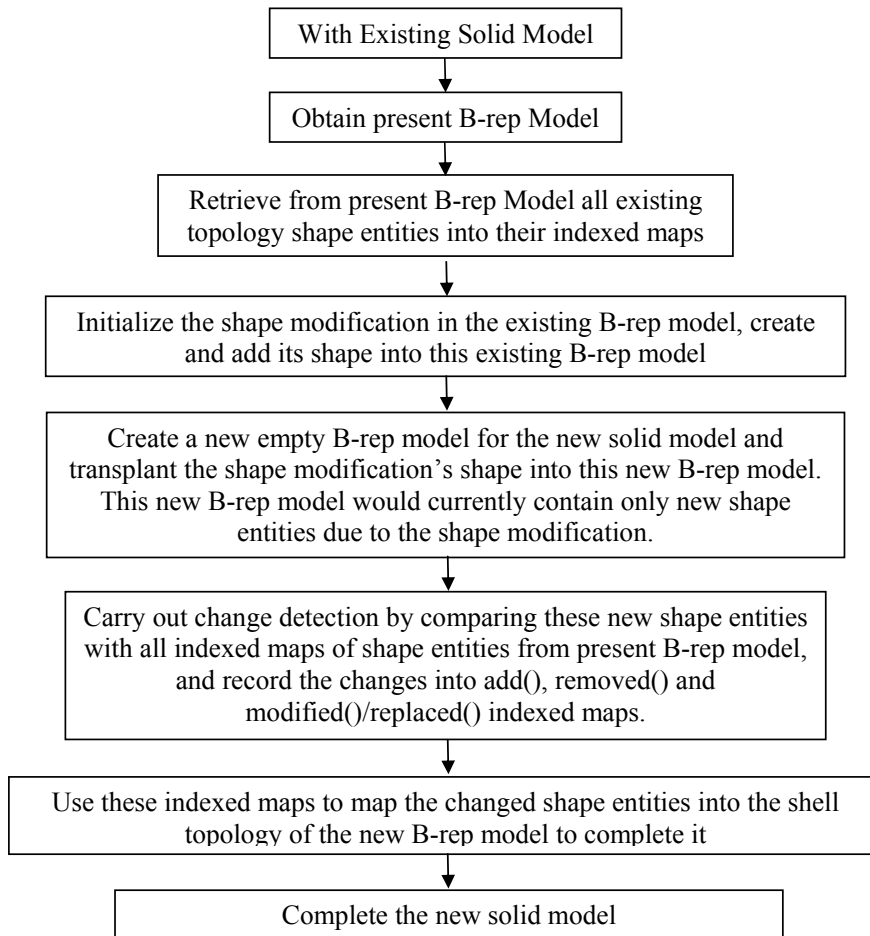


Figure 5.20: Sequence of steps to carry out shape modification with change detection

The steps indicate the necessary B-rep related tasks to be explicitly performed for change detection to occur and shape modification to be completed. To elaborate on the change detection step and in relation to design synchronization with model compression during design change, Figure 5.21 shows a simplified or high level pseudo-code description of change detection algorithm for face shape entity, the concern is to determine new and replaced faces.

Initialize *oldFaceList*, *newFaceList*, *replacedFaceList*, *removedFaceList*;

Evaluate the topological data structure of B-rep models: *oldModel* and *newModel*, and store all the shapes of type TopoDS_FACE into lists *oldFaceList* and *newFaceList* respectively for change detection processing to follow;

For each Face (*face_n*) in the *oldModel*

 Get a list of modified shapes from current Face (*face_n*) using current MakeShape modified() method call for testing;

 if this list is not empty

 Get the new Face (*face_m*) from the list;

 Bind new to old Face /* to be used when constructing a shell */;

 Add *face_n* and *face_m* to the map of replaceShapeMap;

 Add *face_m* to *replacedFaceList*;

 end if

end for loop

For each Face (*face_n*) in the *oldModel*

 Compare *face_n* with each element in *newFaceList* and *replacedFaceList*;

 if *newFaceList* and *replacedFaceList* do not contain *face_n*

 Add current Face (*face_n*) to *removedFaceList*; /* for all removed faces */

 end if

end for loop

For each Face (*face_m*) in the *newModel*

 Compare *face_m* with each element in *oldFaceList* and *replacedFaceList*;

 if *oldFaceList* and *replacedFaceList* do not contain *face_m*

 Add current Face (*face_m*) to *addedFaceList*; /* for newly generated faces */

 end if

 if *oldFaceList* contains *face_m* and *replacedFaceList* does not

 Make a mapping between current Face (*face_m*) and the same Face in *oldFaceList*;

 end if

end for loop

Figure 5.21: Design Change Detection Algorithm for Face Shape Entity

Basically, after an initialization and retrieval of lists of old and new face shape entities (Figure 5.20), the design change detection is done by evaluating the face indexed map of the new B-rep model where there are new faces which *have not been contained* in the old one. In this way, the newly added faces can be found. Whereas, to find a replacement face, a modified() method call to the shape modification command with an existing face from the old B-rep model as input would evaluate whether there are replacement face(s) corresponding to this existing face. In general, the face indexed map of the old B-rep model can be evaluated to see if there are old faces which have not been included in the new B-rep model. In this way, the removed faces can be detected. Usually, design change or modeling operations that cause this are not local shape modifications, but they can occur to the workpiece such as in subtracting with a large hollow operation onto an existing pocket removing the pocket faces, or making a large through hole onto an existing smaller blind hole removing the latter's cylindrical face and base face.

Similarly, for other topological types, such as vertex, edge, wire, shell and solid, added, removed and modified shapes can also be detected for any shape modification operation. Therefore, with such detection, two lists for added and removed shapes, and two lists for shapes modification (replaced) and shapes mapping can be obtained.

In totality, the collections of add(), modify()/replaced(), remove() when obtained respectively contain the indices to the vertex, edge and face shape entities that have state changes. The resulting changed shape geometries are to be further processed onto internal B-rep index maps with the following procedure: the *removed* shapes can be removed from the old B-rep model; the *added* shapes are in the new B-rep model

and the *replaced* shapes are also mapped into the new B-rep model. The new B-rep model is then initialized in the kernel. The remaining entities that are not design change(s) themselves are then automatically mapped into new sequence numbers or tags by the kernel. They can be obtained with another method call to the kernel to retrieve the information. As a supplementary note, this procedure allows a B-rep delta or change to be captured which can be used to carry out design streaming across product modeler servers to maintain consistency.

5.9 Design Change Synchronization for Application View Update

The above design change detection provides the `added()`, `modified()/replaced()`, `removed()` and `mapped()` face entity information affected by a shape modification process. This information should be used for the purpose of updating application views as in product and say fixture design.

These face shape entities would have been associated with updated face tags which can be used to generate face-based faceted models for local compression in the case of `add()` and `modified()/replaced()` collections. In the case of `mapped()` entities, no compression is needed. On the whole, these face tags are to be used to create new augmented Product Data representations for design change update to the application views to ensure that the Java3D scene graph is consistent with the product model.

It is important to note from Figure 5.19, that `modified()/replaced()` and `mapped()` states have associated with it the old tag and new tag information. This is key to application view update as the old tag would be used in the application view's Java3D scene graph to delete the relevant shape node or replace that shape node with the new

face tag and associated augmented Product Data representations, if necessary after decompression of the faceted data, and other original Product Data representations such as surface normals, snap points, etc.

The augmented Product Data representation schema should now contain the presence of a <REPLACEDTAG> and <MAPPEDTAG> information (Figure 5.22). When the <REPLACEDTAG> and <MAPPEDTAG> tags are empty, nothing needs to be done as the presence of <FACETAG> is adequate to indicate a new face. If they are not empty, the appropriate updates would be carried out. The update of the Java3D scene graph is more obvious for newly added faces – it would just be the presence of <FACETAG> together with the compression. Removed face tags can be communicated directly.

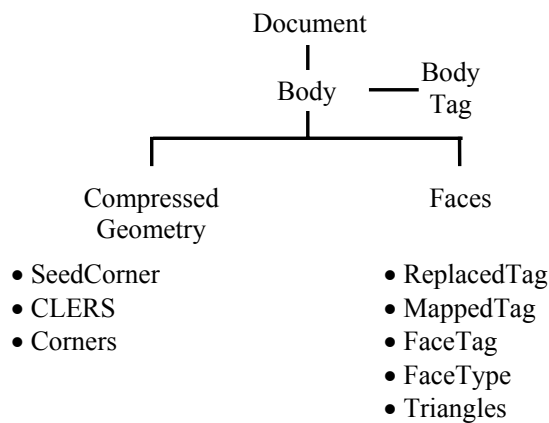


Figure 5.22: Improved Augmented Product data schema to support design change

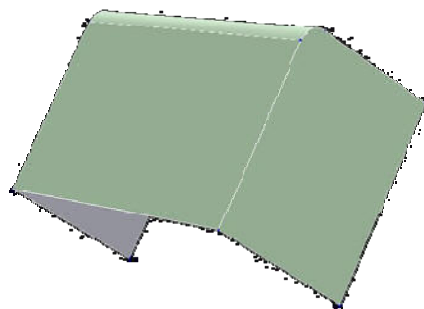


Figure 5.23: The filleted block with new and replaced face shape entities

Figure 5.23 illustrates the presence of new and replaced shape entities associated with a fillet modeling operation.

5.10 Discussion and Summary

To review, the conceptualization and development of the middleware framework and application architecture elements has allowed applications to be developed independently and be seamlessly integrated. This addresses the fundamental problem of distributed collaborative design involving distributed environments containing heterogeneous diverse practices and tools.

The middleware framework has the advantage of integrating applications for collaboration via sharing and accessing a product master model supported by a central geometric modeling server during runtime. Applications that do not have to directly deal with or make design changes to the product model design need only have a product data representation to enable its algorithms to carry out decision making. The realization of the middleware framework and application architecture with interactive rules-based fixture design demonstrates this. However, there is no design synchronization to improve application view updates and account for design changes to the actual product model at the product modeler server.

Design synchronization mechanisms in the middleware framework are thus needed in a manner appropriate to the distribution of functionality and data in distributed environments. Hence, to propose design synchronization for application view updates, two essential techniques have been proposed/evaluated and developed.

Firstly, it has been demonstrated that the role of integrated compression as an enabling technology is feasible and vital to timely, accurate and consistent application view updates across distributed environments. To be able to compact faceted models in association with a product data representation is necessary given that the Internet resources i.e. bandwidth and CPU are shared in a real world of many users and multi-tasking software applications. In particular, it has also been established that handling design change involving face shape entities is not an issue to compression, unlike the assumptions and cumbersome approach of [Wu and Sarma 04].

Secondly, design changes and the fundamental nature of CAD or geometric modeling kernels require application view update to be driven from boundary representation changes in the product model during shape modification itself. This cannot be simply dealt with by developing interactive modeling commands off a library. Without identifying the deltas of design changes involving face shape entities, it is impossible to capture evolving faceted data due to shape modification and support compression to achieve design change updates to application views. The deltas would also have been unavailable for updating product data representations. This has been addressed through improvements to the augmented Product Data representation schema (originally Geometric Data XML schema) for design change.

In particular, distributed collaboration design is featured by collaborative decision-making based on problem solving tools at the application views. With product-process interactions involving design changes, the management of meaningful relationships between application views of the product model is a useful approach. However,

effective design synchronization based on the product model's boundary representation is vital. The next chapter will illustrate design synchronization mechanisms working on a general workpiece object as well as one related to fixture design. Of special interest is the relationship of these design synchronization mechanisms in design change handling and supporting the feasibility of application relations management in integrated product-process interactions.

With an integrated approach, the eventual interest is that of investigating into large scale distributed collaborative design problems. Notably, the entire delta of a design change is used in constructing a new B-rep to complete a design change internally. This can be leveraged to carry out design streaming across product modeler servers to consistently update all product models even though they are physically distributed and different. This is concluded in the next chapter. In this manner, multiple coordination protocols or strategies can be introduced to the computing environment for conflict resolution and what-ifs from a multiple design objective viewpoint for large scale distributed collaborative design.

Chapter 6

Design Synchronization for Collaborative Decision Making

This chapter aims to integrate the developments in the thesis to demonstrate and emphasize design synchronization for early collaborative decision-making in product-process interactions. In terms of design change detection and application view update, it follows the previous chapter with a more detailed and general illustrations of managing product data integrity and consistency. It also provides more effective results about model compression due to design change. A fixture re-design case study is used to focus on early design synchronization of product-process interactions due to design change detection and update. However, for effective design synchronization, a critique and discussion on the current ARM functionality is necessary for collaborative decision-making to be more effective. The key idea is that application views do not have to have complete problem solving nor need to perform unnecessary problem solving given early design change detection and update. This discussion ultimately serves to highlight the importance of appropriate product modeller-driven design synchronization mechanisms to carry out early or timely collaborative decision-making.

6.1 Introduction

Distributed application views are necessary in today's collaboration viewpoint of product-process interactions in fragmented value chains. They facilitate possibly

proprietary applications and problem solving methods belonging to dispersed users or companies. These users would need to share and work on actual product models in a synchronized manner, and to ensure that the product designs that they deal with are co-ordinated in a timely consistent manner to cope with design changes. For this purpose, early design change detection and update is crucial to avoid costly re-import of product models, loss of updated face tags, and unnecessary efforts in problem solving that could exacerbate inconsistencies and increase product development and manufacturing costs.

6.2 Design Change Detection and Update

The delta of a design change can be described as follows:

5. New faces generated:- new tags, new surfaces & facet data,
6. Faces modified/replaced:- new tags, modified surfaces and resulting facet data,
7. Faces unmodified and mapped:- new tags, same surfaces and facet data, and
8. Faces removed:- deleted surfaces with associated facet data

It is pertinent to subsequently understand this in greater details with regards to the problem of persistency and consistency in boundary representations. Persistency has been an issue in making sure that shape entities can always be referred to without having to rely on static file formats. These reference tags are emitted whenever a boundary representation is created when the file format is read into a CAD system or geometric modelling kernel. The key issue is that as a boundary representation is evaluated during a design change, these reference tags are mapped and updated as a form of internal housekeeping. Consistency is the issue of making sure that such reference tags as they are used must be referring to the same shape entities across all

applications. For a single product modelling server or product model, as long as there is design change detection and update, all applications would be able to use reference tags consistently during runtime.

Design change detection is illustrated here in a more general way compared to the Chapter 5 (Figures 6.1-6.4). The illustrations demonstrate design change detection in terms of the B-rep processing in three phases: before, during and after a design change. A different design change or shape modification is used.

Figure 6.1 shows an arm case workpiece of average shape complexity and topology. A fillet modelling operation as a design change is to be applied to the edge selected and highlighted. Noticeable is the shape topology around the selected edge to its left involving multiple affected shape entities. As such, the inside cylindrical face is actually composed of two cylindrical surfaces even though a designer or engineer would have logically perceived or construed as one smooth surface.

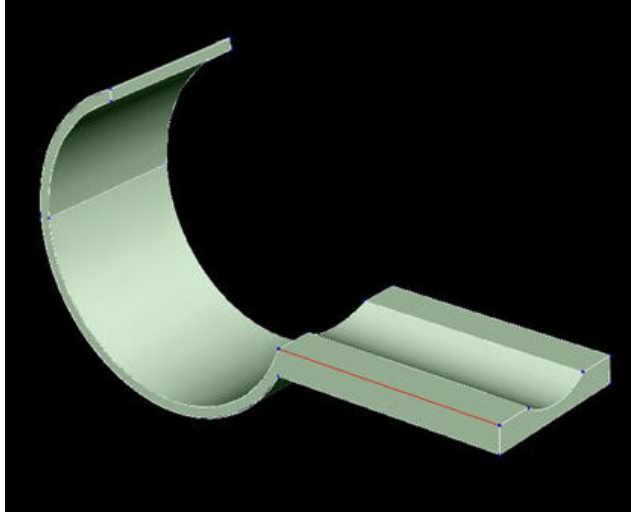


Figure 6.1: An Arm Case Workpiece

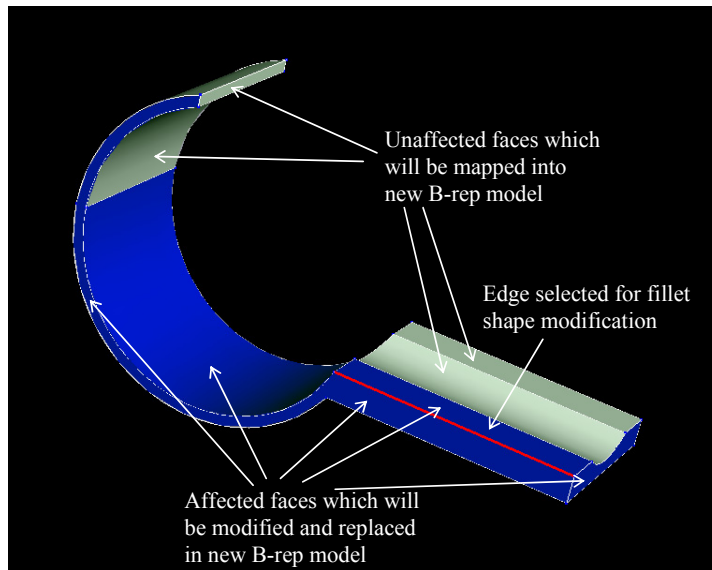


Figure 6.2: Highlighted Affected Faces in old B-rep Model before Design Change

Figure 6.2 shows the workpiece represented as an old B-rep model before shape modification. The key affected and unaffected face shapes due to the shape modification are described. Of interest now is that one cylindrical surface would be unaffected by design change. All of these would be indicated by the design change detection during shape modification (Figure 6.3). This is possible as the affected

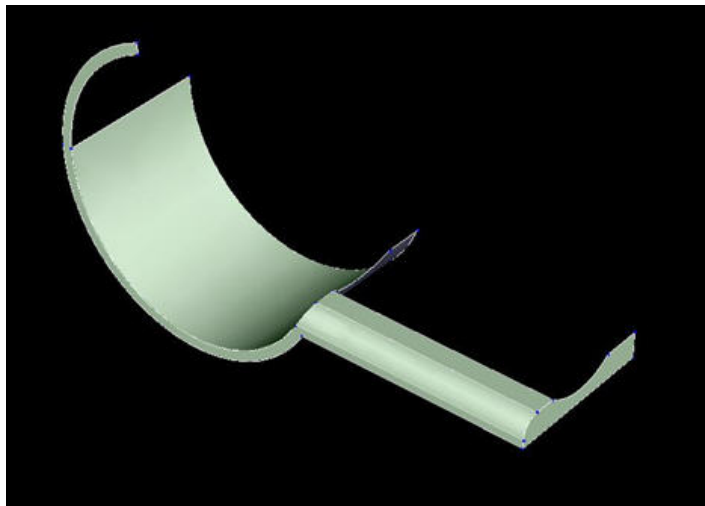


Figure 6.3: Modified/Replaced and New Faces Detected in Design Change

shapes can be indicated within the boundary representation for display. The mentioned cylindrical surface is not shown here as it is not affected at all by the fillet modelling operation. Figure 6.3 would also indicate that only the faceted models of these face shape entities need to be effectively generated for compression and update to application views.

Figure 6.4 shows the design change completed with all shape entities, including the unaffected cylindrical surface, reconstituted and updated.

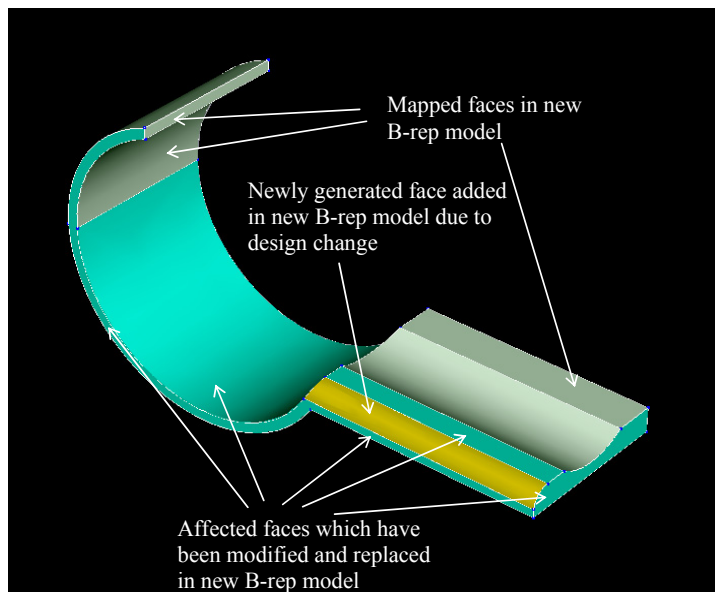


Figure 6.4: Modified/Replaced, New and Mapped Faces in new B-rep Model after Design

6.3 Design Change Synchronization Case Study with Fixture Design

A more detailed illustration of design change involving a workpiece and fixture re-design synchronization is provided to highlight the need for early collaborative decision-making. This comprises the capture of the design change's delta information

and its roles in compression and augmented product data representation in application view updates. Figure 6.5 shows the workpiece.

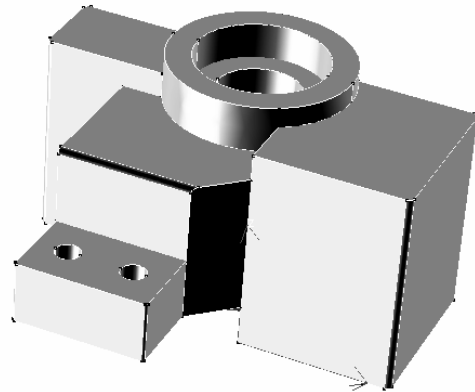


Figure 6.5: Workpiece before Design Change in Product Design Application View

Figure 6.6 shows the details of full model compression applied to the product model to expedite the application view update. In this case study, the interest is on design synchronization involving design change and its detection and update. Figure 6.7 first illustrates an initial fixture design configuration in process whilst Figure 6.8 shows an early design change applied to the workpiece. Early design change detection has to take place followed by a timely update to the fixture design application view (Figures 6.9-6.10).

Figure 6.9 indicates how all shape entities have been processed during design change and boundary representation re-evaluation. Of interest are the face shape entities that have been added, removed and especially, modified/replaced. These face shape entities have their tags mapped and updated (bold). Unmodified face shape entities' tags are also mapped and updated (bold). For instance, in “fa:22->fa:14”, the top surface adjacent to the cylindrical boss has been modified or trimmed by the design change *i.e.* a step operation. The face tag ‘22’ must be updated as ‘14’ in the

```

Full model compression started!
MeshSize0
XML file name compression.xml
Face ID: 0
mesh created V=34 T=32
Euler: 18.0
the mesh size is 1584bytes
Number of Holes: 1
create compressed mesh vertexcount=35 trianglecount= 66
compress: use Predictor: dcd.jedgebreaker.ParallelogramPredictor@9db0ad
_quantdelta=false
the compressed model size is 996bytes
new (empty) mesh created V=35 T=66
decompress: use Predictor: dcd.jedgebreaker.ParallelogramPredictor@ba679e
decompress connected region, seed=2
Write to file
Face ID: 1
mesh created V=26 T=24
Euler: 14.0
the mesh size is 1200bytes
Number of Holes: 1
create compressed mesh vertexcount=27 trianglecount= 50
compress: use Predictor: dcd.jedgebreaker.ParallelogramPredictor@1e8b671
_quantdelta=false
the compressed model size is 772bytes
new (empty) mesh created V=27 T=50
decompress: use Predictor: dcd.jedgebreaker.ParallelogramPredictor@121dcac
decompress connected region, seed=2
Write to file
Face ID: 2
mesh created V=26 T=24
Euler: 14.0
the mesh size is 1200bytes
Number of Holes: 1
create compressed mesh vertexcount=27 trianglecount= 50
compress: use Predictor: dcd.jedgebreaker.ParallelogramPredictor@1ed620
_quantdelta=false
the compressed model size is 772bytes
new (empty) mesh created V=27 T=50
decompress: use Predictor: dcd.jedgebreaker.ParallelogramPredictor@7be687
decompress connected region, seed=2
Write to file

```

Figure 6.6: Typical Output of Model Compression of Workpiece

augmented product representation as well as its faceted model deleted in the Java3D scene graph of all application views. The new faceted model of face tag '14' would also be generated and compressed for all application views (Figure 6.10).

Of specific interest is that old face tag '8' has been modified and replaced by face tag '16', together with new face tag '10' added. This information generated directly from

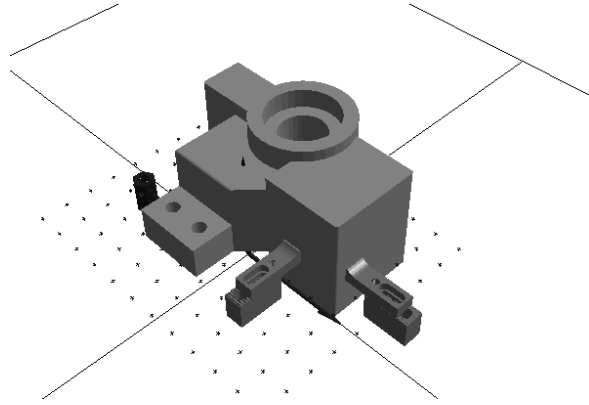


Figure 6.7: An Initial Fixture Configuration in Application View before Design Change

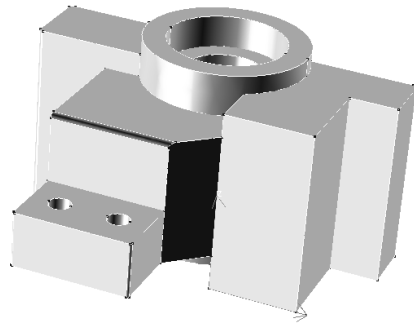


Figure 6.8: Workpiece after Design Change in the Product Design Application View

<p>Added Shapes: ve:38 ve:39 ve:52 ve:53 ve:63 ve:64 ed:57 ed:73 ed:74 ed:87 ed:88 ed:89 ed:97 fa:10 fa:15</p> <p>Removed Shapes: ve:32 ve:33 ve:54 ve:55 ed:39 ed:74 ed:80 ed:81 fa:23</p> <p>Replaced Shapes: ed:38->ed:56 ed:40 -->ed:54 ed:73-->ed:72 ed:76-->ed:86 fa:8- ->fa:16 fa:21-->fa:11 fa:22-->fa:14 fa:29-->fa:39</p>
--

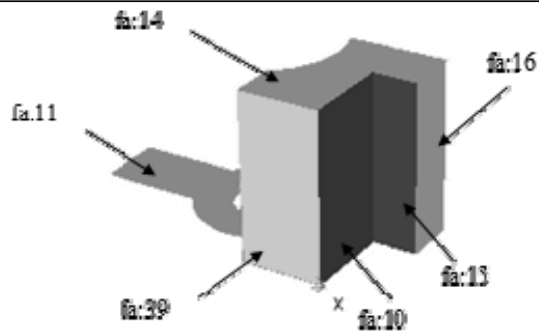


Figure 6.9: Captured Face Shape Entities in Design Change Detection

design change detection is clear and explicit compared to presuming or assuming that

old face tag '8' has been modified and replaced by face tags '10', '16' or even '15'. It is noted that the compression for faces '10', '15' and '16' result in a slightly larger data size, compared to the other larger faces such as '14', '39' and '11'. It is known that model compression works more efficiently for shapes of greater complexity. Nonetheless, it should be noted that with design change detection, only affected face shape entities need to be involved in application view updates. In this way, the updates are considered to be design change-driven.

Affected Face ID: 10	the mesh size is 144bytes	the compressed model size is 156bytes
Affected Face ID: 11	the mesh size is 3264bytes	the compressed model size is 1976bytes
Affected Face ID: 14	the mesh size is 984bytes	the compressed model size is 632bytes
Affected Face ID: 15	the mesh size is 144bytes	the compressed model size is 156bytes
Affected Face ID: 16	the mesh size is 144bytes	the compressed model size is 156bytes
Affected Face ID: 39	the mesh size is 312bytes	the compressed model size is 240bytes

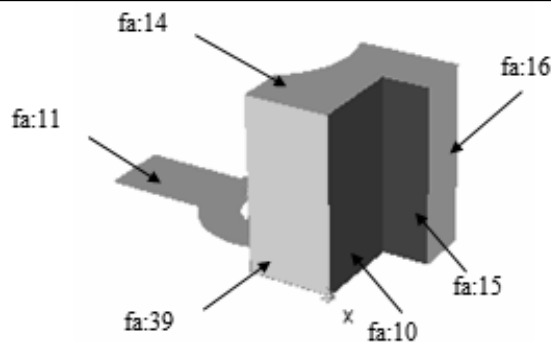


Figure 6.10: Captured Face Shape Entities for Design Change Update through Compression

With the fixture design application view updated, fixture re-design can proceed with the problem solving approach needed, *e.g.* interactive editing, using rules; or applying a flexible fixture re-design methodology or using automated techniques such as

Genetic Algorithms [Mervyn 04]. Figures 6.11-6.12 show the fixture application view before and after a design change update. In this context, the early selection of a locator element has to be reconsidered given new face tag '10'. Figure 6.12 shows the adjustment needed or determined by the appropriate reasoning in the fixture design application view.

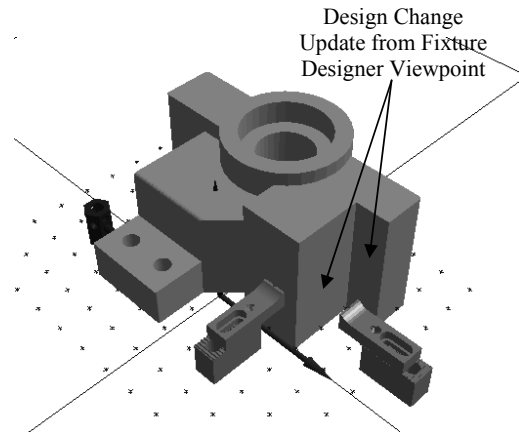


Figure 6.11: Fixture Design with Design Change Update on Application View

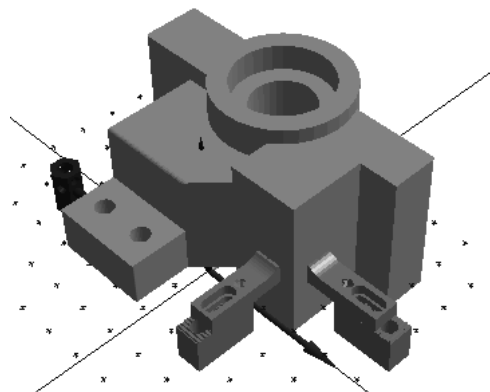


Figure 6.12: Fixture Re-Design Completed on Application View

Of interest is that this case study illustrates an early but incomplete fixture-in-process given that there is no downward clamping force yet applied from the top to the base of the fixture. If a machining operation of a tool cutting across the top is to be considered, a top clamp would be prudent and necessary. It cannot be safely assumed that the cutting tool can provide the downward clamping force as it is moving and will

likely cause vibration. A fixture analysis or simulation might have shown that the initial fixture selection is valid but this only means that design change detection and update to the simulation application view is necessary to either terminate or ignore the simulation session in a fuller collaborative product design and development environment. Essentially, coordination and management protocols in the design concurrency context would be needed. At any rate, early design change detection and timely update to application views is an important capability to provide opportunities to other application views to make their decisions as early as possible.

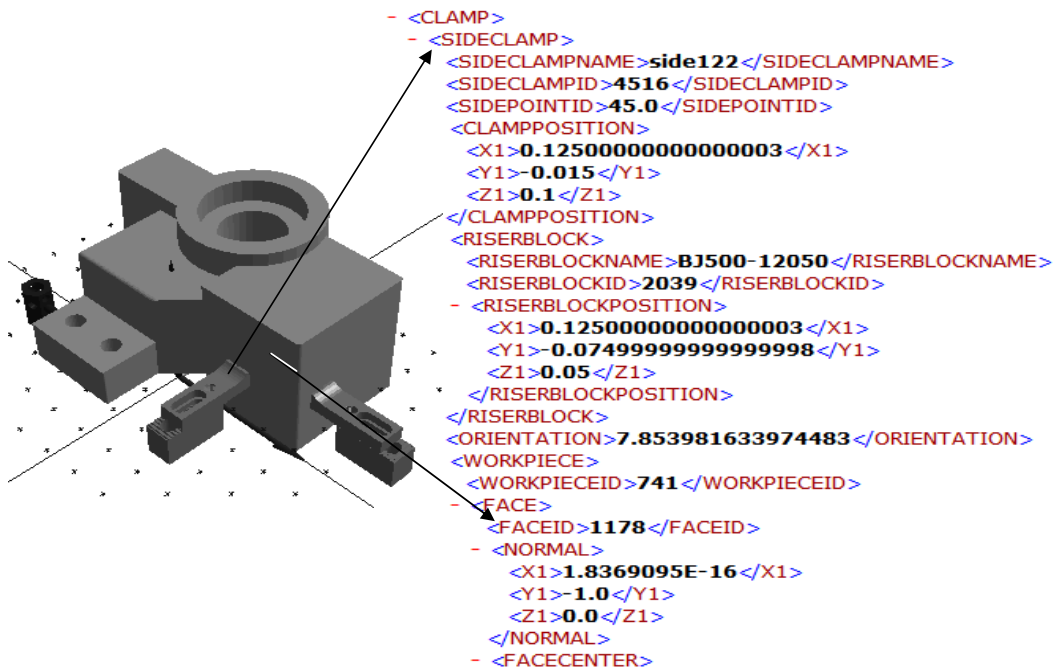


Figure 6.13: Fixture Design Representation with Face Tag Association

At this point, it is important to note that face tags are used in a fixture design XML representation to associate with fixture elements, *e.g.* in Figure 6.13. This association is meaningful as it captures fixture relationships with the workpiece as can be seen in the <SIDECLAMP> </SIDECLAMP> tags encapsulating the <FACE></FACE> tags. However, when these face tags are not properly synchronized from the product

model to the fixture design application view after a design change, then the resulting fixture representation due to fixture re-design would not be correct. The key is to note that re-importing the workpiece is not a solution as that would emit a new boundary representation with new tags. This is further discussed in the context of application relations management approach in the next section. Figure 6.14 shows the runtime command output of design change detection.

```

Added Shape Entities:
ve:38 ve:39 ve:52 ve:53 ve:63 ve:64 ed:57 ed:73 ed:74 ed:87 ed:88 ed:89 ed:97 fa:10 fa:15
Removed Shape Entities:
ve:32 ve:33 ve:54 ve:55 ed:39 ed:74 ed:80 ed:81 fa:23
Replaced Shape Entities:
ed:38->ed:56 ed:40->ed:54 ed:73->ed:72 ed:76->ed:86 fa:8->fa:18 fa:21->fa:21 fa:22->fa:24 fa:29-
>fa:32
Shape Entity Mapping:
ve:0->ve:0 ve:1->ve:1 ve:2->ve:2 ve:3->ve:3 ve:19->ve:4 ve:20->ve:5 ve:21->ve:6 ve:22->ve:7 ve:23-
>ve:8 ve:24->ve:9 ve:25->ve:10 ve:18->ve:11 ve:13->ve:12 ve:8->ve:13 ve:9->ve:14 ve:10->ve:15
ve:11->ve:16 ve:12->ve:17 ve:14->ve:18 ve:15->ve:19 ve:16->ve:20 ve:17->ve:21 ve:7->ve:22 ve:4-
>ve:23 ve:5->ve:24 ve:6->ve:25 ve:26->ve:26 ve:27->ve:27 ve:28->ve:28 ve:35->ve:29 ve:36->ve:30
ve:37->ve:31 ve:38->ve:32 ve:30->ve:33 ve:39->ve:34 ve:31->ve:35 ve:29->ve:36 ve:34->ve:37 ve:60-
>ve:40 ve:61->ve:41 ve:62->ve:42 ve:59->ve:43 ve:40->ve:44 ve:48->ve:45 ve:47->ve:46 ve:49->ve:47
ve:50->ve:48 ve:51->ve:49 ve:52->ve:50 ve:53->ve:51 ve:45->ve:54 ve:46->ve:55 ve:41->ve:56 ve:42-
>ve:57 ve:43->ve:58 ve:44->ve:59 ve:58->ve:60 ve:57->ve:61 ve:56->ve:62 ve:65->ve:65 ve:66->ve:66
ve:63->ve:67 ve:64->ve:68 ve:67->ve:69 ve:68->ve:70
ed:0->ed:0 ed:1->ed:1 ed:2->ed:2 ed:3->ed:3 ed:22->ed:4 ed:23->ed:5 ed:24->ed:6 ed:25->ed:7 ed:26-
>ed:8 ed:27->ed:9 ed:28->ed:10 ed:20->ed:11 ed:29->ed:12 ed:19->ed:13 ed:21->ed:14 ed:9->ed:15
ed:10->ed:16 ed:11->ed:17 ed:12->ed:18 ed:13->ed:19 ed:14->ed:20 ed:15->ed:21 ed:16->ed:22 ed:17-
>ed:23 ed:18->ed:24 ed:7->ed:25 ed:4->ed:26 ed:5->ed:27 ed:6->ed:28 ed:8->ed:29 ed:52->ed:30
ed:53->ed:31 ed:50->ed:32 ed:51->ed:33 ed:57->ed:34 ed:31->ed:35 ed:32->ed:36 ed:34->ed:37 ed:47-
>ed:38 ed:49->ed:39 ed:30->ed:40 ed:43->ed:41 ed:44->ed:42 ed:45->ed:43 ed:46->ed:44 ed:37->ed:45
ed:48->ed:46 ed:56->ed:47 ed:54->ed:48 ed:55->ed:49 ed:42->ed:50 ed:33->ed:51 ed:35->ed:52 ed:36-
>ed:53 ed:41->ed:55 ed:84->ed:58 ed:85->ed:59 ed:86->ed:60 ed:82->ed:61 ed:59->ed:62 ed:87->ed:63
ed:83->ed:64 ed:75->ed:65 ed:67->ed:66 ed:68->ed:67 ed:69->ed:68 ed:70->ed:69 ed:71->ed:70 ed:72-
>ed:71 ed:65->ed:75 ed:66->ed:76 ed:58->ed:77 ed:60->ed:78 ed:61->ed:79 ed:62->ed:80 ed:63->ed:81
ed:64->ed:82 ed:79->ed:83 ed:78->ed:84 ed:77->ed:85 ed:95->ed:90 ed:90->ed:91 ed:96->ed:92 ed:88-
>ed:93 ed:89->ed:94 ed:94->ed:95 ed:93->ed:96 ed:97->ed:98 ed:98->ed:99 ed:99->ed:100 ed:100-
>ed:101 ed:91->ed:102 ed:92->ed:103 ed:104->ed:104 ed:102->ed:105 ed:103->ed:106 ed:101->ed:107
fa:0->fa:0 fa:4->fa:1 fa:3->fa:2 fa:2->fa:3 fa:1->fa:4 fa:12->fa:5 fa:17->fa:6 fa:11->fa:7 fa:15->fa:8
fa:16->fa:9 fa:18->fa:10 fa:10->fa:11 fa:5->fa:12 fa:9->fa:13 fa:14->fa:14 fa:13->fa:15 fa:7->fa:16
fa:6->fa:17 fa:25->fa:19 fa:24->fa:20 fa:19->fa:22 fa:20->fa:23 fa:33->fa:26 fa:30->fa:27 fa:34->fa:28
fa:26->fa:29 fa:31->fa:30 fa:28->fa:31 fa:32->fa:34 fa:35->fa:35 fa:27->fa:36 fa:38->fa:37 fa:37->fa:38
fa:36->fa:39

```

Figure 6.14: Typical Output of Design Change Detection of Affected Shape Entities

6.4 Design Synchronization with Application Relations Management

In the present ARM approach to synchronizing applications for collaborative decision-making in IPPD, the product model is normally set up by a designer to

support application relations by initializing or depositing the model into an ARM object. A deposit action is said depend on the Geometric Data XML file or product data representation to retrieve the relevant information such as tags to update the ARM object for managing relationships.

These relations associate the product model hosted at the geometric modelling server with applications and their functional relations at the level of face shape entities. Each face tag in the ARM object contains a status option defined as “changed” or “unchanged” to highlight design change. When this product model is shared with a process application view, say a fixture design, a face in the product model is then set up to have the application relation function of say, “Locating Face” associated with the same face tag. Faces are referenced by face tags under a body tag, all available from the emitted boundary representation of the product model during runtime at the product modeller server and updated into Geometric Data XML file or product data representation.

When design change is carried out with shape modification in the product model, synchronizing applications with application relations management is carried out. The designer activates the ARM object which notifies the affected applications based on the relations set up earlier with those faces which are now changed. Each affected application view then retrieves the Geometric Data XML or product data representation of the modified product model. At the same time, the notification is said to determine the faces affected by the shape modification, but this is not described although the same product model is being shared from the product modeller server.

The present synchronization approach for application view update requires the entire geometric data XML or product data representation, including the faceted model to be used after design change has occurred. On the part of faceted model data, this would be ineffective and less than timely to carry out application view without employing compression together with the augmented product data representation, and by extension design change-based on updates.

It is now known that with design change, shape entities do have their tags updated or re-defined as the boundary representation is re-emitted and evaluated during runtime. Shape entities during design change are actually in one of the following states: new, replaced, deleted or mapped, all of which need to be taken into account in application view update as mentioned above.

So on the part of design change handling in ARM, it is not clear how the update of the entire geometric data XML can be used to ensure that design change can be detected based on face tags before and after the design change. It is noted in the case study demonstrating application relationship management in dealing with design change in fixture design that the workpiece was reported to be reloaded into the product modeller server.

Also, the 'changed' and 'unchanged' options for each face in the ARM object seem to be incomplete as 'changed' could mean the states of new, modified/replaced and mapped, technically speaking. Also, after re-evaluation, such faces in the ARM object would become invalid as their tag references do not correspond with those updated in

the boundary representation. As such, these tag references cannot be relied upon for tracking.

So it is now known that even with a runtime B-rep model, the problem of persistency occurs with design change as all tag references are revised when the B-rep is evaluated. This causes inconsistency throughout all product-process modelling and interactions. It is impossible to detect design changes correctly without an appropriate B-rep processing technique to discover these up-to-date tags that reference all shape entities. Likewise, it is also impossible to carry out design synchronization with application relations for collaborative decision-making as these relations are associated with faces, just as in the fixture design representation.

The current ARM notification mechanism is said to be able to determine faces affected by design change but it is not clear how this determination is carried out to discover faces modified or deleted as mentioned in the case study. Anyway, as indicated above, inconsistency would have set in from the changes in the boundary representation.

A design change detection and update mechanism would have automatically updated the Geometric Data XML or augmented product data representation beforehand to ensure that all application views have consistent references. Otherwise, relying on a product modelling server to just update the entire Geometric Data XML file would be cumbersome during design changes. It is also not sufficient as all reference tags are updated during boundary representation evaluation and it is important to directly know their correspondence with previous tags. With design change detection and

update, all application relations can be generally and automatically updated at least with the detailed states of modified/replaced, and removed, if not the states of new and unmodified. This would not require an application view, such as fixture design, to be responsible for determining such changes in state.

6.5 Summary

In summary, design synchronization cannot be completely fulfilled in a distributed collaborative design environment without appropriate design change detection within the boundary representation of the product model and the associated updates to application views. In this sense, design synchronization must be ‘driven’ from the product modeller itself so that for instance, application relations management can be carried out.

Dispersed companies often collaborate in an enterprise and have their own product modeller servers producing product master models. These companies acting as customers often provide these product master models to other suppliers of services down the value chain such as conventional process planning which determine intermediate product models relative to downstream fixture design, so on and so forth.

These suppliers may be given access to their customers’ product modeller servers. More likely, they may have a copy of the product master model hosted on their servers to create intermediate models for fixture design suppliers.

This kind of environment can be described as large scale distributed collaborative design in the value chain. In the context of suppliers having to use their product

modeller servers, another form of design synchronization can take place across the product modeller servers from each customer to his immediate suppliers to update the suppliers' product master models to keep them consistent with the customers, as if they are all one.

The assumption is that both customer and suppliers have agreed to start with the same base shape for product design. Essentially, this is called design streaming and it is possible since the B-rep design change detection technique makes aware the shape entities that are new, modified and replaced, removed, and mapped, that can also be topologically reconstructed on a supplier's product modeller server.

Chapter 7

Conclusions and Recommendations

The research conducted in this thesis has been focused on distributed collaborative design characterised by design change demanding design synchronization across distributed environments.

Facilitating distributed collaborative design requires the following issues to be addressed:

1. Underlying middleware framework and application architecture
2. Appropriate distribution of functionality and data
3. Design synchronization with design change detection and update

The middleware framework and application architecture is the foundation for the development of a distributed collaborative design computing environment. Within this environment, there has to be a distribution of functionality and data appropriate to the product design and development domain. Such a distribution guides the implementation of application architecture elements as application views, product modeller servers, and product models and data representation, and their respective functional and data issues and requirements. For instance, application views are not just necessary interactive visualizations of product models and process applications. They support the necessary functionality of the application, such as in fixture design by rules. As another example, product modelling and data representations are guided by the need to resolve compatibility problems to have seamless and flexible integration

to be useful middleware. Also, product data representations are understood to be without an accompanying boundary representation intentionally. This is to permit a process application to sufficiently act on the product model without needing a runtime boundary representation that can only be generated by an accompanying standalone CAD or geometric kernel system. As mentioned before, this arrangement would cause a proliferation of product model versions and static CAD files which would complicate the consistency issue of managing distributed collaborative design.

In distributed environments where product-process interactions have to take place and design changes are encountered, synchronization is needed for timely, accurate and consistent updates. Ultimately, application view updates must be facilitated for early collaborative decision-making to be feasible. The most challenging aspect of design change involves shape modifications which affect the integrity and consistency of product model information that application views rely on to carry out collaborative decision-making. Timely and accurate design change detection and update is vital to ensuring that application views have the opportunity to respond and collaborate through their various problem-solving tools.

The contributions of this thesis are as follows:

- Conceptualisation and development of the middleware framework and application architecture that is based on an appropriate distribution of functionality and data to design a distributive design environment. This is exemplified by enabling a remote product modeller central server(s) and an interactive fixture design application view. Issues addressed include resolving compatibility and integration problems in heterogeneous environments with the

role of reusable object-oriented Java classes as middleware for communication and access into a product model, and effective product data representation for an application like fixture design to be carried out.

- Design synchronization in distributed environments for timely, accurate and consistent application view update with integrated model compression of faceted models and augmented Product Data representation. This includes face-based compression, paving the way for design change detection and update.

In particular, design changes involve newly generated, modified/replaced and removed surfaces. It is thus not necessary to engage the entire product model geometry. These faceted surface models with sufficient complexity due to design change would be usefully compressed for application view updates. Removed surfaces must have their faceted models in the application view deleted.

- Design synchronization for application views to have consistent references to the product model during design change. Design change detection captures shape entity changes in the boundary representation of the product model. Design change *delta* from runtime boundary representation evaluation includes shape entity addition, modification/replacement, removal and mapping of all reference tags. The appropriate face shape entity changes must be updated into the application view's product data structure, *i.e.* Java3D scene graph. Accordingly, new, added, modified/replaced face shape entities and their reference tags mapping must be updated inside the scene graph that contains previous unmapped tags. In this manner, consistent and associated references

are available to application views during product-process interactions and collaborative decision-making. Hence the general approach for application relations and their management would require proper design change updates for effective application representations, information models and problem solving.

The following are some recommendations for future research and improvements:

- Integrate application relations management with design change detection and update at the product modeller server for design synchronization and consistency.
- In the application architecture approach, application views can be extended to product simulation which is integral to product development and manufacturing. Such simulations usually require a finite element model to be developed from the product model. Since distributed collaborative design is characterised by design changes, simulations can be rapidly carried out if change detection and update can be integrated from product design to planning, if not directly to simulation via the finite element model. Product design affecting workpiece and tooling design as in fixture element selection and hence fixture analysis is a case to demonstrate this. Therefore research into design change integrated with and re-defining the simulation model should be explored.
- For large-scale distributed collaborative design in value chains, product modeller servers are bound to prevail between customers and suppliers or companies collaborating together, as it is impossible to have one single master server. Research into the real time (incremental) streaming of design changes across these (heterogeneous) product modeller servers should be explored.

REFERENCES

BIDARRA, R., and BRONSVOORT, W. F., Semantic Feature Modeling, 2000, Computer-Aided Design 32 (2000) 201–225.

BLATECKY, A; WEST, A; SPADA, M., 2002, Middleware – The New Frontier. EDUCAUSE Review, Jul-Aug, 25-35.

BRONSVOORT, W. F., and NOORT, A., Multiple View Feature Modeling for Integral Modeling, 2004, Computer-Aided Design 36 (2004) 929-946.

BIDARRA, R.; VAN DEN BERG, E. and BRONSVOORT, W. F., 2001, Collaborative Feature Modeling, 2001. Proceedings of DETC'01 ASME Design Engineering Technical Conferences, September 9-12, 2001, Pittsburgh, Pennsylvania.

BIDARRA, R; KRANENDONK, N.; NOORT, A. and BRONSVOORT, W.F., 2002, [A collaborative framework for integrated part and assembly modeling](#). Journal of Computing and Information Science in Engineering 2(4): 256-264.

BOK, S.H., SENTHIL, K.A., WONG, Y.S., NEE, A.Y.C., 2004, Model Compression for design synchronization within distributed environments, Proceedings of the International Conference of CAD and Applications, May 24–28, Pattaya Beach, Thailand, 2004(a), pp. 127–136 (also published in the special issue of CAD, 2005, CAD'04, International CAD Conference and Exhibition, May 24-28, 2004.)

BRONSVOORT, W. F. and JANSEN, F. W., Multi-view feature modeling for design and assembly. In *Advances in Feature Based Manufacturing, Manufacturing Research and Technology*, ed. J. J. Shah, M. Mantylä and D. S. Nau. Elsevier Science, 1994, pp. 315–330.

CHAN, S., WONG, M. and NG, V., 1999, Collaborative solid modeling on the WWW. *Proceedings of the 1999 ACM Symposium on Applied Computing*, San Antonio, CA, pp. 598–602.

CIGNONI, P.; MONTANI, C.; and SCOPIGNO, R., 1998, A Comparison of Mesh Simplification Algorithms, *Computers & Graphics*, vol. 22, no. 1, 1998, pp. 37-54.

DE KRAKER, K. J., DOHMEN, M. and BRONSVOORT, W. F., 1995, Multiple-way feature conversion to support concurrent engineering. In *3rd Symp. On Solid Modeling and Applications*, ed. C. Hoffman and J. Rossignac. Salt Lake City, UT, 1995, pp. 105–114.

DE KRAKER, K. J., DOHMEN, M. and BRONSVOORT, W. F., Feature validation and conversion. In *CAD Systems Development*, ed. D. Roller and P. Brunet. Springer-Verlag, Heidelberg, 1997.

DE KRAKER, K. J., DOHMEN, M. and BRONSVOORT, W. F., Maintaining multiple views in feature modeling. In *4th Symp. on Solid Modeling and Applications*. ACM Press, 1997, pp. 123–130.

DEERING, M., 1995, Geometry compression, Proceedings of SIGGRAPH '95, pp 13–22.

EDGEBREAKER, J., Triangle Mesh Compression Software in Java. <http://www.igd.fhg.de/coors/JEdgebreaker/jedgebreaker.html>

EL-SANA, J.; and VARSHNEY, A., 1997, “Controlled Simplification of Genus for Polygonal Models,” *Proc. IEEE Visualization 97*, IEEE CS Press, Los Alamitos, Calif., 1997, pp. 403-412.

SCHROEDER, W., 1997, “A Topology-Modifying Progressive Decimation Algorithm,” *Proc. IEEE Visualization 97*, IEEE CS Press, Los Alamitos, Calif., 1997, pp. 205-212.

HAN, J.H., and REQUICHA, A.A.G., 1998, Modeler-independent feature recognition in a distributed environment, *Computer-Aided Design*, 30(6), pp 453–63.

HOFFMAN, C.M.; and JOAN-ARINYO, R., 1998, CAD and the product master model, *Computer-Aided Design*, Vol. 30, pp. 905-919.

HOFFMAN, C.M.; and JOAN-ARINYO, R., 2000, Distributed maintenance of multiple product views, *Computer-Aided Design*, Vol. 32, pp. 421-431.

HOPPE, H., 1996, Progressive Meshes, *Computer Graphics (Proc. SIGGRAPH 96)*, vol. 30, ACM Press, New York, 1996, pp. 99-108.

HUANG, G.Q., MAK, K.L., 1999, Web-based collaborative conceptual design, *Journal of Engineering Design*, 10(2), pp 183-194.

HUANG, G.Q., MAK, K.L., 2003, *Internet applications in product design and manufacturing*, Springer-Verlag.

LI, W.D., QIU, Z.M., 2006, State of the art technologies and methodologies for collaborative product development systems, *International Journal of Production Research*, vol. 44, no. 13, 2525–2559.

LINDSTROM, P. et al., 1996, Real-Time, Continuous Level of Detail Rendering of Height Fields, *Computer Graphics (Proc. SIGGRAPH 96)*, vol. 30, ACM Press, New York, 1996, pp. 109-118.

MAROPOULOS, P. G., 1999, Aggregate product and process modeling for the welding of complex fabrications. *Annals of the CIRP*, 48(1), 401–404.

MAROPOULOS, P. G., 2003, Digital enterprise technology: defining perspectives and research priorities. *Int. J. Computer Integrated Manufacturing*, 16(7-8), 467–478.

MARTINO, T., FALCIDIENO, B., and HABINGER, S., 1998, Design and Engineering Process Integration Through a Multiple View Intermediate Modeler in a Distributed Object-Oriented System Environment. *Computer-Aided Design*; 30(6): 437–452.

MERVYN, F., 2001, Development of a Virtual Assembly Evaluation Environment, NUS Mechanical and Production Engineering. BE Thesis.

MERVYN, F., SENTHIL, K.A., BOK; S.H., and NEE; A.Y.C., 2003, Development of an Internet-enabled interactive fixture design system, *Computer-Aided Design* 35(10), pp 945-957.

MERVYN, F., SENTHIL, K.A., BOK; S.H., and NEE; A.Y.C., 2004, Developing distributed applications for integrated product and process design, *Computer-Aided Design* 36:(8), pp 679-689.

MERVYN, F., SENTHIL, K.A., BOK; S.H., and NEE; A.Y.C., 2004, Design change synchronization in a distributed environment for integrated product and process design, CAD'04, International CAD Conference and Exhibition, May 24-28, 2004.

MERVYN, F., PhD Thesis, "Integrated Product and Process Design Across An Extended Enterprise: An Implementation in Fixture Design" Mechanical Engineering Department, National University of Singapore, 2004.

NAM, T.J. and WRIGHT, D.K., 1998, COLLIDE: A shared 3D workspace for CAD. Proceedings of the 1998 Conference on Network Entities, Leeds.

<http://interaction.brunel.ac.uk/~dtpgtjn/neties98/nam.pdf>

NIST Planning Report, 1999, Interoperability Cost Analysis of the U.S. Automotive Supply Chain, Prepared by Research Technology Institute, Gaithersburg, MD.

PAHNG, G.D.F., BAE, S., and WALLACE, D., 1998, A web-based collaborative design modeling environment, Proceedings of the IEEE Workshops on Enabling Technologies Infrastructure for Collaborative Enterprises (WET ICE'98), pp 161-167.

PTC Product Lifecycle Management, <http://www.ptc.com/products/plm/index.htm>

RATNAPU, K.K., 2001, Web Based CAD System, NUS Mechanical and Production Engineering. MEng Thesis.

REGLI, W. and PRATT, M., What are feature interactions? In: McCarthy JM, editor. CD-ROM Proceedings of the 1996 ASME Computers in Engineering Conference, 19–22 August, Irvine, CA, USA, New York: ASME, 1996.

ROSSIGNAC, J. 1997, The 3D revolution: CAD access for all, Int Conf Shape Model Appl, Aizu-Wakamatsu, Japan, Silver Spring, MD: IEEE Computer Society Press; pp. 64–70.

ROSSIGNAC, J., 1999, Edgebreaker: connectivity compression for triangle meshes. IEEE Trans Vis Comput Graph, 5(1):47–61.

Roy, U., and Kodkani, S.S., Product modeling within the framework of the World Wide Web, IIE Transactions 1999; 31(7), pp 667–677.

SCHANTZ, R.E. and SCHMIDT, D.C., 2001, Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications, Encyclopedia of Software Engineering, edited by John Marciniak and George Telecki, Wiley and Sons.

SCHLENOFF, C., GNUNINGER, M., TISSOT, F., VALOIS, J., LUBELL, J., and LEE, J., 2000, The process specification language (PSL): Overview and version 1 specification. NISTIR 6459, NIST.

SENTHIL, K.A.; BOK, S.H.; TAN, B.C.; KUMAR, R.K.; NEE, A.Y.C., 2000, The development of an internet enabled interactive automated fixture design system. Proceedings of the 7th International Conference on Mechatronics, edited by Charles Ume, pp. 1-7, 6-8 Sep 2000, Atlanta, U.S.A.

SHIKHARE, D., 2001, State of the Art in Geometry Compression, National Centre for Software Technology, India (dinesh@ncst.ernet.in)

SHYAMSUNDAR, N., AND GADH, R., 2001, Internet-based collaborative product design with assembly features and virtual design spaces. Computer-Aided Design, 33:637–51.

SHYAMSUNDAR, N., AND GADH, R., 2002, Collaborative virtual prototyping of product assemblies over the Internet. Computer-Aided Design, 34:755–68.

SOHLENIUS, G., 1992, Concurrent engineering. Annals of CIRP, 41(2), 645–655.

SPATIAL, 2002, ACIS 3D Modeling Kernel, Version 7.0. Spatial Technology Inc., Westminster, CO. <http://www.spatial.com>

STORK, A. and JASNOCH, U., 1997, A collaborative engineering environment. Proceedings of TeamCAD '97 Workshop on Collaborative Design, Atlanta, GA, pp. 25–33.

Sun Microsystems, 1999, Java3D API specification, <http://java.sun.com/products/javamedia/3D>.

UGS PLM Solutions, 2004, <http://www.eds.com/products/plm/>

WANG, L.H.; SHEN, W.M.; XIE, H.; NEELAMKAVIL, J.; and PARDASANI, A., 2002, Collaborative conceptual design – state of the art and future trends, Computer-Aided Design, Vol. 34, pp 981-966.

WU DI, SARMA R., 2004, The incremental editing of faceted models in an integrated design environment, Computer-Aided Design, Vol. 36, pp 821 - 833.

XIE, S.Q., TU, P.L., AITCHISON, D., DUNLOP, R., and ZHOU, Z.D., A WWW-based integrated product development platform for sheet metal parts intelligent concurrent design and manufacturing. Int J Prod Res 2001; 39:3829–52.

PUBLICATIONS ARISING FROM THIS THESIS

Journals

Bok S H, Senthil kumar A, Wong Y S, Nee A.Y.C, Model compression for design synchronization within distributed environments, *Computer-Aided Design and Applications*. Vol. 1, no. 1-4, pp. 67-73. 2004.

Mervyn F, Senthil kumar A, Nee, A Y C, Design change synchronization in a distributed environment for integrated product and process design, *Computer-Aided Design and Applications*. Vol. 1, no. 1-4, pp. 43-52. 2004.

Mervyn F, Senthil kumar A, Bok S H, Nee A Y C, Developing distributed applications for integrated product and process design, *Computer Aided Design*, 2004:36(8), 679-689.

Mervyn F, Senthil kumar A, Bok S H, Nee A Y C, Development of an Internet-enabled interactive fixture design system, *Computer Aided Design*, 2003:35(10), 945-957.

Conferences

Mervyn F, Senthil kumar A, Bok S H and Nee A Y C, Development of a Reference Enterprise Model for Fixture Design Information Support in Integrated Manufacturing, 2003 ASME International Mechanical Engineering Congress and Exposition, *MED 14*, pp. 259-266, Nov 15-21, Washington D C, USA

Mervyn F, Senthil kumar A, Bok S H and Nee A Y C, Internet-enabled smart interactive fixture design, 2002 Japan-USA Symposium on Flexible Automation, July 14-19, Hiroshima, Japan

Senthil kumar, A, Tan B C, Bok S H, Kiran R K and Nee A Y C, The development of an Internet-enabled interactive fixture design system, 7th Mechatronics Forum and International Conference and Mechatronics Education Workshop, CD-ROM ISBN 0-08-043703 6, Sep 2000, Georgia Institute of Technology. USA