Developing distributed applications for integrated product and process design

F. Mervyn, A. Senthil Kumar*, S.H. Bok, A.Y.C Nee

Department of Mechanical Engineering, The National University of Singapore, 10 Kent Ridge Crescent, Singapore 119260, Singapore

Received in revised form 14 July 2003; accepted 16 July 2003

Abstract

A heterogeneous computing environment characterizes today’s manufacturing situation. This is a stumbling block for the efficient implementation of manufacturing concepts such as integrated product and process design (IPPD). A computing environment for IPPD would require the seamless integration of the various product and process design software systems. The exchange of information between these systems should be efficient, compatible and synchronous. This article presents an approach for developing distributed manufacturing applications that are compatible and synchronized and thus, able to support IPPD. The approach involves the use of a common manufacturing application ‘middleware’, which is distributed between a central geometric modelling server and application clients. The portability of the middleware is ensured through the use of Java for code portability and XML for data portability. The compatible product model problem is solved through the use of common data structures developed using reusable application client classes. Efficient transfer of product data is proposed using compressed model information embedded in a product data XML schema. Synchronization of design changes among all applications is achieved through the creation of relationships on an Application Relationship Manager.

© 2003 Elsevier Ltd. All rights reserved.

Keywords: Distributed application development; Integrated product and process design; Middleware; XML

1. Introduction

The evolving economic situation in the world has caused manufacturing companies to reformulate their business strategies. To remain competitive today, many companies have adopted strategies to concentrate on their core competencies. This has resulted in a manufacturing environment where the different stages of a product development process are carried out by geographically and temporally distributed companies. This situation is often characterized by a heterogeneous computing environment.

Further, present day customer demands point towards a need for greater customization and for shorter product lead times. Global competition to meet this demand has resulted in the introduction of various manufacturing concepts, practices and technologies. Integrated Product and Process Development (IPPD) [1] is one such practice that aims to reduce product lead-time and cost as well as improve product quality. IPPD has been defined as a management process that integrates all activities from product concept through production/field support, using a multifunctional team, to simultaneously optimize the product and its manufacturing and sustainment processes to meet cost and performance objectives. The central notion to this concept is that product quality and user satisfaction can best be achieved by the integrated concurrent design of the product and its processes. However, the heterogeneous computing environment characteristic of the distributed manufacturing situation is a stumbling block for the implementation of IPPD.

An ideal computing environment for IPPD should emulate a round-table discussion where product and process designers determine the final design of a part and the processes needed to manufacture and support the part. Consequently, a criterion for a computing environment for IPPD would be seamless integration of the various product and process design software systems. Seamless integration involves compatibility and synchronization of the information accessed by applications. However, the development of product and process design applications that are to be

* Corresponding author. Tel.: +65-6874-6800; fax: +65-6779-1459. E-mail address: mpeask@nus.edu.sg (A. Senthil Kumar).
seamlessly integrated is a tall order in today’s dynamic product development environment. As companies collaborate with an array of different partners depending on the constraints of a situation, the required interfaces with other applications are not known during the development of an application. Application developers are often burdened with these interfacing problems. Ideally, application developers should concentrate on solving domain related problems without these imposed burdens.

Recent research in application development has looked into means of developing distributed applications in an effort towards a pervasive computing environment in manufacturing. However, present distributed manufacturing applications have been mainly developed in isolation without due consideration of integrating the various applications. It is the aim of this paper to provide a solution to this problem through an approach of developing distributed manufacturing applications that can be seamlessly integrated and thus, able to support IPPD.

This paper is organized as follows. Section 2 discusses the related research in developing integrated computing environments for product and process design. Section 3 describes the details of an approach for developing integrated and synchronized manufacturing applications. Section 4 discusses prototype development of applications based on the approach. Section 5 presents an illustrative case study and Section 6 concludes the paper.

2. Related research

Manufacturing application development is carried out mainly in two ways. One is to develop applications based on a CAD system’s application programming Interface (API). Another is to build applications directly over solid modelling kernels.

Early applications were deployed on computers coupled together with the CAD package or modelling kernel. Several approaches were developed in creating an integrated environment for product and process design based on these standalone systems.

One approach is the use of standard file formats such as STEP and IGES for CAD models located at central databases. Roy and Kodkani [2] proposed the use of a translator to convert CAD models into VRML based models which can then be viewed over the WWW. The VRML models are stored in an existing product data repository. The translator resides on a main central server and can be accessed remotely by a designer. Xie et al. [3] 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. The geometry of the part was represented by STEP files. Wang and Zhang [4] developed an integrated CAD/CAPP/CAM system that is supported by an Internet/Intranet network and TCP/IP protocol and is based on central databases to support collaborative product development. A feature based product definition model was used.

The drawback in the use of standard file formats is that the approach provides a static interface to applications [5]. In IPPD, various design changes occur and manufacturing systems would need to retrieve the entire CAD model each time a design is changed. As the changed file is not linked to the previous file, the approach is not susceptible to incremental process design. As such, this approach is inefficient for a distributed environment and provides limited synchronization.

Hoffman and Joan-Arinyo [6] describe another approach based on a client/server architecture for a product master model that unites CAD systems with downstream application processes for different views that are part of the design process. They presented a practical approach to synchronize geometric data contributed by a CAD system and data from other application programs through the creation of associations.

Recent research in application development has looked into means of developing distributed applications. Several researchers have proposed the use of a central geometric modelling server as a means of developing these distributed applications. Han and Requicha [5] discussed an approach that provides transparent access to diverse solid modellers for applications in a distributed environment. Solid modellers were augmented with software wrappers to provide a uniform API. Their system encompasses a feature-based design system, a central geometric modelling server, an automatic feature recognizer and a graphics renderer. The central geometric modelling server stores the Brep model of a designed part. When a design change occurs, the design system communicates the change to the feature recognition system. The drawback of this approach is that the design system needs to know the different applications that are accessing the designed part to communicate changes so that a synchronized model can be accessed by all systems.

Shyamsundar and Gadh [7,8] proposed a client-server based architecture for collaborative virtual prototyping of product assemblies over the Internet. A polygonized representation of the part was used for visualization and an Internet-centric, compact assembly representation was also developed. A solid modeller was employed as an application server to remove the complexity of installation and maintenance of the solid modeller from the client. In their system, 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.

Bidarra et al. [9] developed a web-based collaborative feature modelling system known as webSPIFF. It is based on a client-server architecture where the server coordinates the collaborative session, maintains the shared model and
makes use of a multiple-view feature modelling kernel [10]. All views are kept consistent by feature conversion.

Present approaches in developing distributed applications concentrate on specific applications. Integration of these distributed product and process design systems has not been effectively dealt with. As mentioned earlier, effective integration of product and process design systems requires compatible and synchronous information exchange. Although the work reported in Ref. [9] allows the views of other applications to be synchronized through feature conversion, the drawback of the approach is that only views that are implemented on the modelling kernel are kept consistent. Thus, it is not a general solution to synchronizing the various product and process design systems. In this paper, a general approach for the development of distributed manufacturing applications is discussed such that applications can be developed independently, but easily integrated, interfaced and synchronized. The proposed approach also makes application development easier through the use of reusable client classes. We adopt a similar approach to the synchronization of design changes through the use of reusable client classes. We adopt a similar approach to the synchronization of design changes among all applications, as discussed in Ref. [6], through the creation of relationships with the product model. However, based on our approach we avoid the problems of persistent naming in CAD systems and the need for geometry certificates as discussed in Ref. [6]. Various considerations regarding efficient data transfer over the Internet have been taken into account in arriving at the architecture. The approach is also susceptible for incremental process design.

3. Seamless integration through a common manufacturing application middleware

Our approach, in developing distributed applications that are seamlessly integrated, is based on the use of a common manufacturing application ‘middleware’. The middleware is a set of layers of software components that sit between solid modelling kernels and manufacturing applications as shown in Fig. 1. The layers of the middleware are distributed between application clients and a central server. The solid modeller 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 modeller interface is responsible for interfacing the server end with solid modelling kernels. The information model layer contains information from the solid modeller in a neutral form. It could also contain other information deposited by application clients. The communications infrastructure allows applications to make remote function calls to the server. The reusable application classes are a group of reusable classes that applications use for development. The following sections discuss the details of the middleware implementation.

3.1. Central modelling server architecture

The central modelling server plays a key role in providing a dynamic interface to applications accessing and manipulating the product model. This section discusses the architecture of the server, shown in Fig. 2. The server was implemented in Java and consists of the following components: (i) Java Remote Method Invocation (RMI) Interface (ii) Server classes (iii) Java Native Interface (iv) Parasolid Modelling Kernel (v) Apache HTTP Server (vi) MySQL server and (vii) the Edgebreaker algorithm [11].

The communications infrastructure has been developed using Java RMI. The Java RMI interface describes the remote methods, which clients will use to interact with the server. Java RMI allows Java objects running on one computer to call a method running on another computer as if that method was part of the same program running on the same computer. Although other distributed object technologies such as CORBA are available, Java RMI has been adopted to form the communications infrastructure as in the present system, the reusable application client classes and the server classes have been developed using Java. As such, there is no need for an Interface Definition Language (IDL) such as CORBA.

In the present system, two main interfaces have been implemented: Modelling functions and the Applications Relationship Manager. The modelling functions interface allows application clients to make remote calls to a solid modelling kernel, giving application clients the ability to manipulate and interrogate the product model. This allows the development of applications without the installation of a modelling kernel on every machine the application is to be run. The Applications Relationship Manager allows applications to build relationships with the product model. Synchronization of product models among all applications is carried out through the Applications Relationship Manager. The Applications Relationship Manager will be discussed in detail in Section 3.3.

The Server classes implement the methods described in the RMI Interface. Clients interact with objects of the Server classes to carry out the actual manipulation and interrogation of the product models. In the developed
system, the Parasolid modelling kernel has been utilized to carry out the manipulation of product models. As the modelling kernel is written in the C programming language, a JNI is needed to utilize the modelling functions. The JNI allows Java code that runs within a Java Virtual Machine (JVM) to operate with applications and libraries written in other languages. As such, the Server classes also take the role of declaring the native methods of the Parasolid modelling kernel.

The information models layer is implemented using XML files stored in an Apache HTTP Server and through a MySQL relational database server. Various product information models have been discussed in the literature to integrate product design and other domains. These include feature based [12] and function based [13] product information models. In this work, the authors concentrate on describing products at the geometric level. When a modelling function is carried out on the server, geometric information about the model is written to a Product Data XML file and stored in the Apache HTTP server. The Product Data XML file contains information for clients to visualize and interrogate a part. The schema will be discussed in detail in Section 3.2. The MySQL server is used to archive product models in Parasolid’s native format.

In the present system, visualization of products on the client is based on a tessellated representation of the part. Hardware-assisted rasterization is particularly effective at rendering tessellated triangles [14], but a vast amount of storage is required to represent triangular meshes. This results in inefficient transmission of product models for visualization over the Internet. For efficient transmission, the Edgebreaker algorithm [11] for 3D model compression, developed at the GVU Centre at Georgia Institute of Technology, has been utilized in this system. A Java implementation of the Edgebreaker compression algorithm [15] was utilized.

3.2. Model compression and product data XML schema

In this section, we discuss the details of model compression and the Product Data XML Schema. The product data XML schema should be compact for efficient data transfer over the Internet. This will reduce the amount of data to be transferred over the Internet. Furthermore, this criterion is more critical when XML is used for data exchange. When an XML file is parsed, a Document Object Model (DOM) is created to represent the data. The creation of a DOM can be expensive due to the required memory allocation. By making the XML schema compact, the time and memory required to parse the XML can be reduced, improving the efficiency of the system.
Model compression using the Edgebreaker algorithm [11] provides a compact representation for the visualization of the model. However, the Edgebreaker algorithm was mainly developed for visualization of 3D models and as such the compressed data format does not contain modelling information such as face tags. This information is important as clients interact with the modelling kernel based on this information. For example, in a feature-based design system a user has to specify a face on which a feature is to be created. To solve this problem, we augment the compressed data format for 3D models with modelling information, namely, face tags and face types. The sequence of events for arriving at the Product Data XML Schema is shown in Fig. 3.

In this sequence of events, when a modeling operation is carried out, a tessellated mesh of the model is created by invoking a function call on Parasolid. The mesh data from Parasolid is then formatted as required for the Edgebreaker algorithm to work. The 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. We refer the reader to [11] for the details of the Edgebreaker algorithm.

To create a link between the compressed geometry data and the modelling information, we associate the triangles of the tessellated model to the faces that they belong to and store this information in the Product Data XML schema. The structure of the Product Data XML schema is shown in Fig. 4.

The present XML schema only contains information for visualizing and interrogating a model. However, the extensibility of XML allows the inclusion of other information such as features and function. Tags in XML follow a hierarchical structure. The root tag of an XML file is always (DOCUMENT). In the product data schema, each part is identified by a (BODYTAG). The schema is divided into two groups of data, the (COMPRESSIONGEOMETRY) tag contains data on the compression of the tessellated model and the (FACE) tag contains modelling information at the face level. A (FACETAG) is present to identify the various faces of the body. (FACETYPE) provides information on the type of the face, for example, cylindrical, plane and spherical. The (TRIANGLES) tag contains the indices of the triangles that belong to a face. This way, the modelling information is not lost due to the compression as associations are still maintained and a compact representation is still obtained. An example XML schema for a cube is shown in Fig. 5.

The effectiveness of the Edgebreaker algorithm in reducing the data required for visualizing a 3D model has been validated by comparing the sizes of XML files with compressed geometry format and uncompressed mesh data. The results of our experiments are presented in Table 1 for some basic primitive models. The experimental results show a significant compression of the data required, proving the effectiveness of using the Edgebreaker algorithm for model compression.

3.3. Applications relationship manager

The Applications Relationship Manager serves two main roles. First, it ensures that all applications access a consistent product model. Secondly, it creates relationships between the different applications through the product model. These roles are facilitated through the creation of relationships by clients on the faces of a product model through the Applications Relationship Manager.

The Application Relationships Manager contains several methods which facilitate the creation of relationships by clients. These methods are remotely accessed by applications through Java RMI. The remote methods of the Applications Relationship Manager are as follows:

- public void deposit_model (int bodytag): This method is to be called by an application to make a product model ready for creating relationships. The method is normally called by a design client. The input to the method is the body tag of the product model. The method subsequently retrieves data from the appropriate Product Data XML file and updates tables created in the MySQL database for managing relationships. The structure of the information model for managing of relationships is shown in Fig. 6.

The Product List table contains a list of the various models that relationships can be built on. The models
are identified by their body tags. Each product model is divided into faces, which are identified by face tags and are stored in the Face List. The Face List table also contains data on the status of the face. This attribute could take one of two values, changed or unchanged. Each face in turn is connected to a group of client relationships stored in the Client Relationships table. The Client Relationships table stores data on the IP address of the client that made the relationship, the client type, the type of relationship and any comments that an application client would like other applications to take note of. Restrictions are not made on application clients to specify certain values for client type and type of relationship. Typical values of client type could be ‘process planning client’ or ‘assembly modelling client’. Type of relationship refers to how the application relates to the face. For example, an assembly modelling client could describe the type of relationship as a ‘mating face’.

public Boolean create_relationship (RelationshipInfo info): This method allows an application client to associate itself with faces of the model. The input to this method is a class RelationshipInfo which is a part of the reusable application classes. The definition of class RelationshipInfo is shown in Fig. 7.

When the method is called, it subsequently checks if the model has been deposited for creating relationships. If so, it will update the Client Relationships table with the information from the Relation-
shipInfo class. If a relationship is created successfully, the method returns a TRUE Boolean value to the calling method. If a relationship could not be made, it returns FALSE.

- public RelationshipInfo [] query_relationship (int facetag): This method allows an application client to query information on the client relationships that have been made on a face. This will allow application clients to know which domains will be affected by a design change on a face. In the comments attribute in the RelationshipInfo class, clients could restrict other domains from suggesting changes to be made to that face. The method returns an array of RelationshipInfo objects to the calling method.

- public void transmit_design_change (int bodytag): This method is called by any application which makes a change to the product model. The reusable application classes contain classes for applications to carry out modelling. As such, every application can make or suggest a design change. This capability is useful, as to support IPPD, modifications should be introduced by the application that requires the change [16]. When this method is called, it obtains the IP addresses of all the applications that have associated with the model and triggers an application client method to update the product model. However, before a design change is carried out, it is necessary to discuss the change with all application clients affected. A messaging system [17] based on Java Messaging Service has been developed in this work for the purpose of such discussions. However, it will not be discussed here as it is beyond the scope of this paper.

In the application relationship manager, relationships are made on the face tag of models. Tags are created by modelling kernels to identify the particular entities within a modelling session. Each tag in a session is unique, so that every entity can be identified. However, tags are not consistent across different sessions. If the same entities are loaded into a different modelling session, they are not necessarily assigned the same tags. This results in the lost of the relationships which have been created. This is sometimes referred to as the persistent naming problem. However, in the present approach, once a modelling server is started, it is in fact never shut down and hence a modelling session is always live. This way, the tags always maintain a unique value and the persistent naming problem is avoided.

### 3.4. Reusable client classes

To allow application developers to concentrate on domain related functionality, common reusable client classes have been developed which remove interfacing burdens from an application developer. 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.

The client remote interface class has been defined to facilitate interaction with the server implementation of the interface. This allows applications to call the modelling functions of the modelling kernel and to create associations using the Application Relationship Manager. The client remote interface class also contains a method that the server can call to propagate design changes. A product data XML parser class has been developed that parses the product data XML files and stores the information in developed data structures. The visualization class is responsible for visualization of the product models. The Java3D API has been utilized for visualization in the system. The visualization class retrieves information from the data structures and decompresses the cleris, vertices and handles data into facet data. The facet data is then be sent to Java3D classes for rendering into the Java3D canvas, which will be also be set up by the class.

### 3.5. Discussion

The middleware allows applications to be developed independently and be seamlessly integrated. The uniqueness of the middleware is that it unites all applications in interrogating a product model through the central geometric modelling server. This provides a common point of management of the applications, allowing design changes to be transmitted to all applications at the same time. Thus, the middleware allows synchronization of the applications to be achieved.

Compatibility of product model data is achieved through the reusable application client classes. The product data XML parser parses the Product Data XML schema and stores the data in developed data structures. As these data structures are common to all the applications, compatibility is achieved. What makes this approach different from the use of standard file formats for compatible information exchange is that this approach provides applications with a dynamic interface to the product model. In the use of standard file formats for data exchange, when a design change occurs, the applications have to retrieve a new file, which is not linked to the previous file. For example, a particular face could be identified with a certain tag in the first file and a different tag in the new file. This static interface to the product data is not susceptible for incremental process design. In the present approach, as the identification of the model entities are not changed, it creates a link between the modified model and previous model. These changes could be made sense of to allow incremental process design.
4. Implementation

The proposed architecture has been evaluated through the development of a design client, an assembly evaluation and modelling client and a fixture design client. This section discusses the implementation of these application clients.

The software architecture of the design client is shown in Fig. 8. The client starts the application with the Graphical User Interface class and utilizes the visualization class for setting up the Java3D canvas for rendering of models. The design module comprises of Constructive solid geometry (CSG) [18] and feature based modeling [19] capabilities. The Design Module utilizes the remote reference interface class for calling modelling functions on the modelling kernel. The CSG modelling functions in the present system include all the primitive solid functions (block, sphere, cylinder and prism), Boolean operations (union, subtraction and intersection) and transformation operations (translation and rotation). The feature library consists of the following features: Hole, Slot, Pocket, Pad, Boss, Chamfer, Blend, Offset, Draft, Hollow and Taper. Each main feature has several levels of subclasses, for example class ‘Hole’ has subclasses ‘Through Hole’ and ‘Blind Hole’, which further have subclasses ‘Simple Hole’, ‘Counter-sink Hole’ and ‘Counter-bore Hole’.

The software architecture of the assembly evaluation and modeling client [20] is shown in Fig. 9. The assembly evaluation environment is a design space where designers can verify the assemblability of different parts. To carry out assembly evaluation, the appropriate Product Data XML files for the different parts are loaded onto the virtual environment. The Product Data XML parser and Visualization classes are utilized for this purpose. Assembly evaluation is carried out based on feature interaction. The feature information is derived from the Product Feature XML schema. This information group forms a part of the information layer. However, as mentioned earlier, in this paper we concentrate on describing products using low-level entities and do not discuss the feature information part of the information models layer.

The parts are manipulated upon loading onto the virtual environment such that the features to be assembled come into contact with each other. The VCollide [21] collision detection library from the University of North Carolina has been utilized to determine which faces are in collision. As the VCollide collision detection library was written in the C++ language, a JNI was implemented to make Java function calls to the library. The faces in collision can then be traced to the feature that it belongs to and assembly evaluation is carried out. Please refer to Ref. [20] for details on evaluation of actual assembly. Legal mating features can then be assembled using assembly functions present in

![Fig. 8. Software architecture of design client.](image8)

![Fig. 9. Software architecture of assembly evaluation and modelling client.](image9)

![Fig. 10. Software architecture of fixture design client.](image10)

![Fig. 11. Case study scenario.](image11)
the modelling kernel. The remote reference interface class is utilized for this purpose.

The software implementation of the fixture design client [22] is shown in Fig. 10. The Fixture Design module consists mainly of the interactive fixture design methodology and the fixture design XML file generation and parser classes. The main fixture design methodology starts the interactive fixture design application and guides the user through the design process. The fixture design XML file generation class is responsible for obtaining all the essential data about the completed fixture design and embedding the data in the XML file. The fixture design file parser retrieves information from completed fixture designs for the purpose of regenerating on the client. The details on the fixture design client can be found in Ref. [22].

This section has shown how application clients have been developed based on the common reusable client classes. It has been shown that domain specific functionality has been decoupled from the reusable application classes.

5. Case study

To illustrate the benefits of the proposed architecture, we present a case study of IPPD involving product designers and manufacturers. Four companies are involved in this case study. Company A produces product A, which is made up of five parts. It designs Parts 1–3 but outsources the manufacturing of these parts to Company B. It purchases Part 4 from company C and Part 5 from company D. This scenario is depicted in Fig. 11.

In such a scenario, if every company creates its own legacy system for product and process design, huge compatibility problems would arise. Synchronized integration of activities would be a far-fetched dream. However, using the proposed approach, we show how synchronized integration is possible. It is assumed that all companies have developed applications based on the reusable client classes. Company A runs the central geometric modelling server at its site. Company A designs Part 1 using its own design client and deposits the model in the Application Relationship Manager. The designed part and the corresponding information model set up for relationship management are shown in Fig. 12.

Company B manufactures Part 1 by first casting and then machining the different features. The tolerance of slot A is critical as Part 4 is to be assembled via the slot. The fixture designer of Company B now loads the part into the fixture design client (Fig. 13) they developed and carries out fixture design. The fixture designer decides to use faces A, B and C as locating faces and creates associations with these faces. An example association for a face is as follows:

Client Type = “Fixture Design Client”;
Type of Association = “Locating Face”;
Comments = “Face used to locate workpiece”;

Concurrently, Company C loads Part 1 into their assembly evaluation client to check the assemblability of slot A with a boss of Part 4. Company C deems the assembly as feasible and creates associations with the faces of slot A with the following information:

Client Type = “Assembly Evaluation Client”;
Type of Association = “Assembly”;
Comments = “Face part of a feature used for assembly. Do not change”;

Fig. 12. Designed part and corresponding information model.

Fig. 13. Fixture design client of company B.
Company D also loads Part 1 into their assembly evaluation client (Fig. 14) to check the assembly of Part 5 with Part 1. It deems that assembly is not feasible and requires a slot to be included as shown in Fig. 15. Company D can immediately find out which companies or domains it would affect by suggesting the change through the Application Relationship Manager. In this case, the fixture design of Company B would be affected. Company C, however, is not affected. We assume that the design change is critical and company D makes the change to the model. The Application Relationship Manager then determines all clients that have made associations with the product model and propagates the change. Subsequent action due to the design change of an affected client could be manual or could be automated if algorithms have been developed to automatically deal with changes.

This case study has illustrated how the proposed approach can be used to develop integrated and synchronized applications for IPPD and thus, reduce the product lead-time. An example scenario to explicitly show this, is in a case where the fixture designer is unable to design a fixture based on the present set of modular fixtures and requires a dedicated fixture. A dedicated fixture could take as long as three months to design and manufacture. To reduce the product lead-time, the fixture designer could suggest design changes so that the work is fixturable using modular fixtures. This could significantly reduce the product lead-time.

However, it is recognized that in order to create a truly ‘plug and play’ environment that allows any company to collaborate with another company, a standardized architecture for the middleware is required. Two areas of concern are a standard and neutral interface to solid modelling kernels and a concrete application framework for developing applications. A standard neutral interface to solid modelling kernels would allow applications to be developed in a modelling kernel neutral fashion. Further research will be needed to look into developing a layer of the framework that will allow general method calls that are applicable to different modelling kernels. An application framework will combine the different reusable client classes into a framework that can be specialized to produce custom applications. The framework will not only describe the classes, but will also describe how the classes interact, the interface of the classes and the flow of control between them. This will enhance the reusability of the generic classes, as it would define the common architecture underlying the applications [23].

6. Conclusions

This paper has presented an approach for developing distributed applications that can be developed independently, but easily integrated, interfaced and synchronized. The contributions of this paper are summarized as follows:

- The use of a common manufacturing application middleware was proposed to solve compatibility and synchronization problems between different distributed applications. The use of Java and XML allowed for the portability of code and data on different operating platforms. The common middleware provides all applications with a dynamic interface to the product model.
- Efficient transfer of product data has been proposed using compressed model information embedded in a product data XML schema. The compressed data format has been augmented with modelling information to enable user interaction with the product model at the client end.
- An approach to communicate design changes to all related applications has been developed through the creation of relationships on an Application Relationship Manager. This aids to synchronize design and manufacturing planning processes.
Future research will look into developing the capabilities of the middleware such that it is solid modeler independent. An application framework is also being developed to enhance the reusability of the generic classes.

References