Design Change Synchronization in a Distributed Environment for Integrated Product and Process Design

F. Mervyn[1], A. Senthil Kumar[2] and A. Y. C. Nee[3]

[1]National University of Singapore, engp1618@nus.edu.sg
[2]National University of Singapore, mpeask@nus.edu.sg
[3]National University of Singapore, mpeneeyc@nus.edu.sg

ABSTRACT

This paper deals with the synchronization of application views of product data under design changes. This involves two main aspects, (i) a mechanism for propagating design changes to all the applications and (ii) algorithms for dealing with design changes. The use of a common manufacturing application middleware service, known as the Application Relationship Manager (ARM), is proposed as a mechanism to propagate design changes. Dealing with design changes is a domain-specific task that depends on the formulation of the domain process. How design changes can be dealt with is explored using fixture design as a domain example. The fixture design process is formulated using two different approaches, as a sequential design process and as an evolutionary search process.

Keywords: design change synchronization, integrated product and process design, fixture design

1. INTRODUCTION

Integrated Product and Process Design (IPPD) [12] is a product development concept that aims to reduce product lead-time and cost as well as improve product quality through the integrated concurrent design of the product and the associated processes to realize the product. A major research objective is the development of a computing environment that effectively supports IPPD.

Present day design and manufacturing applications do not support IPPD effectively due to:

- Compatibility problems: In today's product development environment, various different companies collaborate to realize a product. The use of different software products often results in compatibility problems. Studies have shown that compatibility problems had cost manufacturing companies about US$1 billion per annum in the automotive industry alone [10].
- Lack of proper information support: Information exchange is a critical component of a computing environment for IPPD. Downstream applications of the product development process require the right information to carry out their tasks, while upstream applications require feedback information. Although a large amount of research has been conducted on developing information models for different applications, present day commercial applications often do not provide the required information.
- Efficiency problems: Various design changes occur in IPPD if the requirements of other domains are not met. Each time a design change occurs, applications need to retrieve the updated information. Retrieving large data sets, such as traditional CAD files, is time consuming and unproductive.
- Synchronization problems: In the concurrent design of the product and the associated processes, it is necessary that all applications are accessing the correct and updated data. Today's design and manufacturing applications work in isolation and proper mechanisms are not in place for effective propagation of design changes.

Various research efforts have proposed credible solutions to these problems and presented integrated computing environments that can support IPPD to various extents. Cutkosky et al [2] presented a notable work in this regard based on an agent approach. Agents were used to encapsulate already developed engineering tools and agent interaction was based on shared concepts and terminology for communicating knowledge across disciplines. The use of a central repository as a product master model was another approach described by

Hoffman and Joan-Arinyo [5] to create an integrated computing environment. The clients of the master model are domain-specific applications that can deposit and retrieve information from the master model. The master model repository provides mechanisms for maintaining the consistency of the deposited information structures. These efforts were mainly aimed at integrating standalone design and manufacturing applications. Standalone applications are applications that are deployed together with the modeling kernel or CAD systems on individual computers.

With the advent of the Internet and the associated World Wide Web (WWW), we are witnessing a new group of Web-based and Internet-enabled applications being developed. These applications are generally referred to as distributed systems. These applications show promise in achieving a pervasive computing environment for product development; an environment in which the architecture of client computers does not matter anymore. Users can access the applications from any computer and carry out their tasks. However, many of these applications are presently being developed in isolation, without consideration of the integration issues. To prevent a similar integration problem as with standalone systems from arising, we have recently presented an approach for developing these applications such that they can be easily integrated and thus, able to support IPPD [8].

In this paper, we concentrate on the synchronization problem. The synchronization problem in a computing environment for IPPD is different from the synchronization problem in collaborative CAD systems which are primarily based on co-visualization. In collaborative CAD systems, all the CAD clients accessing the product data have a common view of the product data, as no further processing is carried out on the data. Synchronization of the CAD clients accessing the product data is achieved through refreshing all clients in real-time taking into account a controlled sequence of operations carried out by the different CAD clients. In an IPPD environment, downstream applications carry out some form of processing on the product data to derive a view of the product necessary to carry out their tasks. Synchronization involves maintaining the consistency of all application views of the product data. This involves two main aspects, (i) a mechanism for propagating design changes to all the applications and (ii) algorithms for dealing with design changes. In this paper, we present the use of an Application Relationship Manager (ARM) [8] that can be deployed on applications to ensure design changes are propagated synchronously and applications are accessing the correct and updated information. We also discuss how design changes can be dealt with, using fixture design as an example domain.

This paper is organized as follows. Section 2 presents the related research in automatic synchronization of downstream applications when a design is modified. Section 3 presents the background of our approach in developing distributed applications for IPPD. Section 4 describes the ARM. Section 5 discusses how design changes can be dealt with in fixture design and Section 6 concludes the paper.

2. LITERATURE REVIEW

In this section, we review some of the approaches in automatically synchronizing design changes in downstream applications. The feature concept has been instrumental in making sense of product data for various downstream applications. It is thus a viable option to synchronize downstream applications through the use of features.

One means of doing this is through the design by feature approach. The design by feature approach is to design a product based on the downstream definition of features. Wang and Wright [13] developed a feature based CAD system called WebCAD. In their system, a designer designs a product based on negative features. This allows easy analysis on the manufacturability of the part. Also, since the manufacturing features are already defined in the product view, the manufacturing application and the design application are automatically synchronized when a design change is made. A drawback of this approach is that it restricts the product designer by only allowing the use of negative features. Another drawback is that it only synchronizes one view, the manufacturing view in this case.

Another approach to automatically synchronize design changes in downstream applications is the multiple view feature model approach. The multiple view feature model approach generates different views of the product model automatically through feature conversion. de Kraker et al [3] developed a system for feature validation and conversion. Each view has its own feature model, which is validated by maintaining all the constraints in the view. A central cellular model is used to link the different views and carry out the conversion. This also allows changes to be made in any view. A distributed version of this system has also been developed [1]. Jha and Gurumoorthy [6] presented an algorithm for automatically propagating feature modifications across different domains. The automatic propagation is based on an algorithm [7] for extracting multiple feature interpretations of a part across domains. De Martino et al [4] presented the use of an intermediate part model that

is shared among applications to obtain multiple views for applications. In their system, feature recognition techniques are used to derive feature-based models and keep the different feature based descriptions consistent.

Hoffman and Joan-Arinyo [5] represented another approach to propagate design changes through net-shape element association. They presented the use of a master model repository which contains mechanisms where an association can be created and maintained. We presented a similar approach [8] for propagating design changes through the creation of relationships on the geometric elements of a part model. This is facilitated through the use of an ARM. The main advantage of this approach as compared to feature conversion is the ability of the approach to be applied to a wider range of applications. While research has been carried out on the definition of features for different domains, not all applications carry out reasoning based on a feature representation. 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 the part that they access to fixture a part. When a design change occurs, it is relevant to the fixture design application how the face has changed.

In this paper, we show how the ARM can be deployed to create a hierarchy of relationships to support the synchronization of product and process design applications. Dealing with design changes is specific to a domain. In the paper, we use the fixture design domain to show how design changes can be dealt with. We show that dealing with design changes is dependent on how domain processes are formulated.

3. DISTRIBUTED APPLICATIONS FOR IPPD

In this section, we present a brief review of our approach to developing distributed applications for IPPD. Our approach is based on the use of a common manufacturing application middleware.

Middleware is, in general, a set of layers that sit between applications and commonly available hardware and software infrastructure to make it feasible, easier and more cost effective to develop and evolve systems using reusable software [11]. Each layer of the middleware offers services that clients can invoke to perform operations needed to achieve application goals. Schantz and Schmidt [11] decomposed middleware into four kinds, host infrastructure middleware, distribution middleware, common middleware services and domain specific middleware services. Host infrastructure middleware enhances native operating system communication and concurrency mechanisms to create reusable network programming components. Examples include the Sun Java Virtual Machine (JVM) and Microsoft's .NET platform. Distribution middleware extend the capabilities of host infrastructure middleware by defining higher-level distributed programming models. Examples of distribution middleware include CORBA, Java RMI and Microsoft's DCOM. Common middleware services are domain independent services that augment distribution middleware. These include OMG's CORBAservices, Sun's Enterprise Java Beans and Microsoft's .NET Web services. Domain specific middleware are tailored to the requirements of particular domains. Schantz and Schmidt [11] pointed out that domain-specific services are the least mature of middleware layers today. However, they also commented that these middleware services have the most potential to increase system quality and decrease time and effort to develop applications. We envision that middleware technology or distributed object computing technology in general will be instrumental in the future development of design and manufacturing applications. We have thus experimented with the development of a common manufacturing application middleware to solve interoperability problems between the different distributed applications being developed in the manufacturing domain.

Our middleware is implemented in the form as shown in Figure 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.

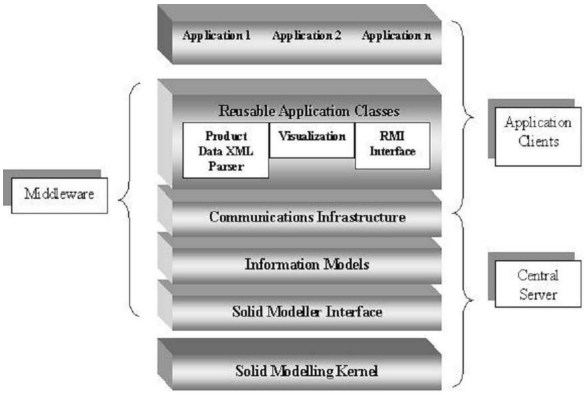

Fig. 1. Framework for developing independent and integrated systems

The middleware offers two key services, ability to make function calls to solid modeling kernels and the ARM. Function calls to the modeling kernel result in the formation of Product Data XML files. These files contain

geometric data of the part and are stored in the server. The schema of the Product Data XML file is shown in Figure 2 and an example XML file for a cube is shown in Figure 3. The reusable application classes allow parsing and visualization of the product data stored in the XML files. The middleware provides applications with a compatible and dynamic interface to product models. Readers are referred to [8] for further details on the middleware implementation. The Application Relationship Manager is crucial in the synchronization of design changes and will be discussed in greater detail in the following section.

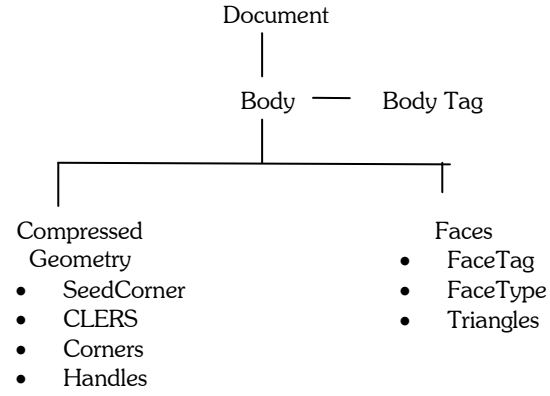

Fig. 2. Product data XML Schema

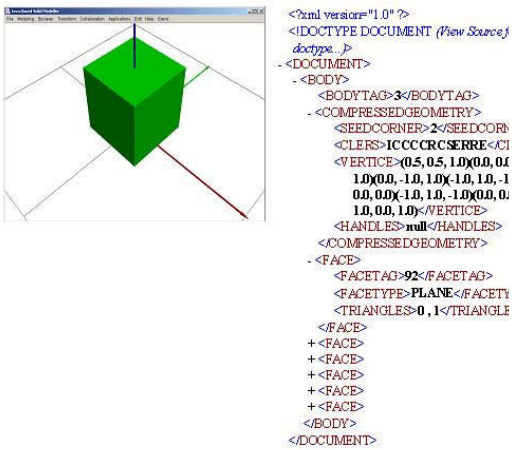

Fig. 3. XML schema for a cube

4. DESIGN CHANGE PROPAGATION

4.1 Description of ARM

The ARM has been implemented using Java RMI and contains two RMI interfaces, a server interface and a client interface. A Java RMI interface describes methods that other applications can access. Applications clients access the server interface to deposit models for relationship creation, create and delete relationships, query relationships and transmit changes. The client interface is accessed by the server to inform all clients that a change has been made.

The server interface provides functionality through the following methods:

- public void deposit_model (int bodytag): This method allows a client to deposit a model in the ARM. This model should already have been created in the modeling kernel. Depositing the model allows other applications to create relationships with the model.
- public boolean create_relationship (RelationshipInfo info) and public boolean delete_relationship (RelationshipInfo info): These methods allow applications to create and delete relationships with the geometric elements of the deposited model. We have defined faces as the only geometric elements in our present implementation. The RelationshipInfo class contains the required information for the method to make the relationship.
- public RelationshipInfo[ ] query_relationship (int facetag): This method allows clients to query the different applications that have created relationships with a particular face. This allows clients to understand how the design variables would affect other applications.
- public void transmit_design_change (int bodytag): When this method is called, the ARM makes a call to all client applications that have created relationships with the product model to inform the clients that a change has been made to the product model.

The client interface contains one method that the server calls to transmit design changes. The input to the method is a list of faces that have been altered due to the design modification. An application client can then make sense of the changes and deal with the changes.

The use of geometric elements to create relationships with and propagate changes is beneficial in two respects. Firstly, geometric elements are generic enough to be applied to a wide range of applications. Secondly, since many downstream applications carry out reasoning based on geometry, relationships with geometric elements are meaningful and allow applications to deal with changes intelligently.

4.2 Deployment of ARM

The ARM is generally deployed in all applications that have interactions with downstream applications

accessing the application data. In this section, we show how a hierarchy of relationships is created through the simple product development environment as shown in Figure 4.

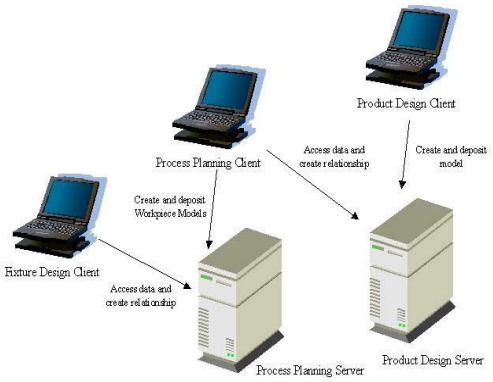

Fig. 4. Example of a product development environment

In this environment, there are two servers, a product design server and a process planning server. Both the servers host the two services offered by the middleware and a geometric modeling kernel. The product design client first creates a part on the product design server using the middleware service to make function calls to the modeling kernel. It also deposits the model in the ARM to allow applications to create relationships. As an example, assume the product design client has deposited the part shown in Figure 5. This part has 16 faces which the downstream applications can create relationships with. The process planning client is able to access the Product Data XML file stored in the product design server and carry out its tasks. We assume the process planning client carries out feature recognition, groups features into setups and determines the sequence of setups. The process planning client first carries out feature recognition and creates relationships with the faces that belong to a feature. This is illustrated in Figure 6.

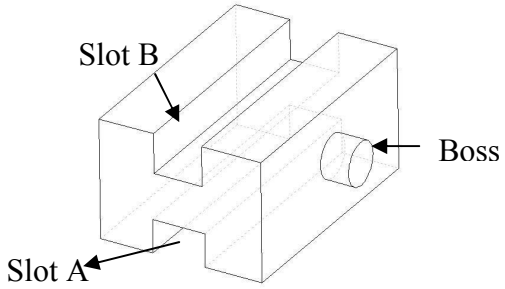

Fig. 5. Example product design

Based on the feature groups and sequence of setups, the process planning client creates the intermediate part/workpiece models of each setup that needs to be fixtured. These intermediate part models are deposited in the process planning server's ARM. The Fixture Design client can now access the data and create relationships with the faces of the intermediate part model. The creation of relationships by the fixture design client is shown in Figure 7.

This section has illustrated how a useful and basic geometric hierarchy of relationships can be created by deploying the ARM on all applications that have downstream applications accessing data. The hierarchy creates a clean separation and distribution between applications that have direct relationships and indirect relationships. This way, if the process planning client makes changes to the intermediate part models, the design changes are only transmitted to the affected applications. The advantage is that the product model does not have to be affected. Also, if the product design is modified, it goes through the process planning client before the change is transmitted to the fixture design client. In this way the transmitted change is relevant to the fixture design client.

5. DEALING WITH DESIGN CHANGES

When a design change occurs, all applications that created relationships with the product model or the intermediate part model are notified of the changes that have occurred. It is then up to the application client to deal with the design change. Ideally, it should not restart its sequence of activities, but deal with it adaptively. Dealing with design changes adaptively depends largely on domain-specific process formulations. A domain usually goes through a series of activities before arriving at a solution for the problem that the domain is trying to solve. An activity is normally dependent on other activities for some form of input to fulfill its goals. In general, if Activity B of a domain depends on Activity A, it is necessary to carry out activity A and then carry out B again to ensure that the factors that Activity B was dependent on are consistent.

In this section, we explore how design changes can be dealt with adaptively using the fixture design domain as an example. We first show how design changes are dealt with in a sequentially interactive fixture design system [9] and then reformulate the fixture design process to remove the sequence of activities through an evolutionary search approach.

5.1 Dealing with design changes in a sequentially interactive fixture design system

In [9], we presented a sequential methodology for interactive fixture design. The general sequence of the activities of the methodology is shown in Figure 8(a).

The sequence begins with the loading of the workpiece followed by choosing the locating, supporting and clamping elements respectively.

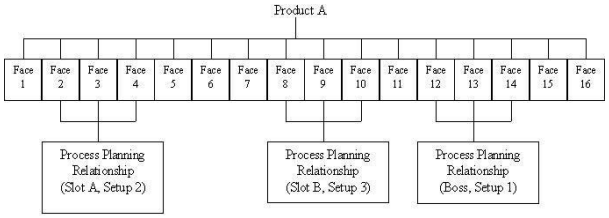

Fig. 6. Relationship created by process planning client

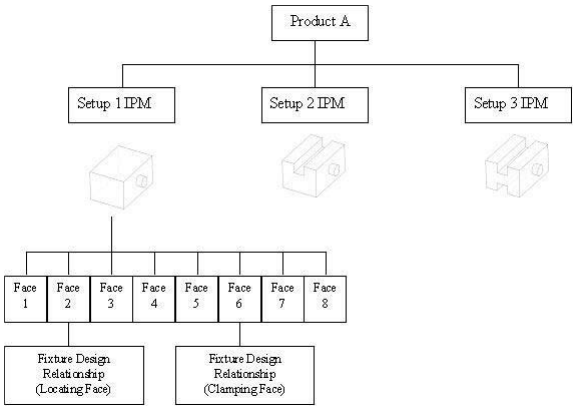

Fig. 7. Relationship created by fixture design client

In the methodology, each activity is dependent on the previous activity as shown in Figure 8(b). For example, the locating elements determine the location of the workpiece on the XY plane. The ability of the supporting and clamping elements to access the necessary faces of the workpiece depends on the workpiece location and thus, on the locating elements. Similarly, the supporting elements determine the height that the workpiece is raised which in turn determine the number of risers needed to raise a clamping element to access a clamping face.

Due to this sequence of activities, design changes in the interactive fixture design system are dealt with based on the methodology shown in Figure 9. The methodology is triggered when a design change is transmitted to the fixture design client. It is applicable in both situations when a fixture design has already been completed or when the fixture design process is ongoing.

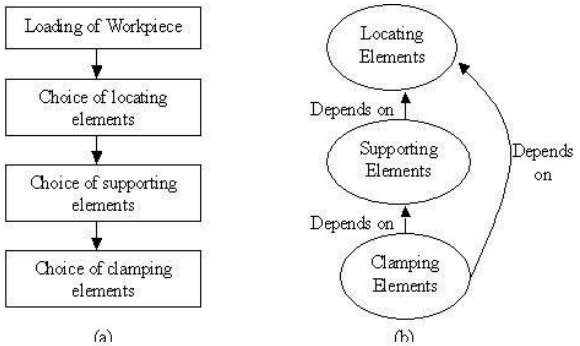

Fig. 8.General sequence of interactive fixture design methodology (a), dependency of choices in interactive fixture design methodology (b)

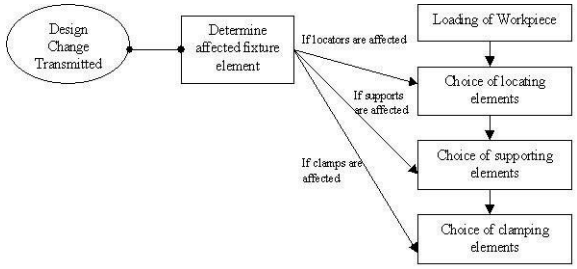

Fig. 9 Dealing with design changes in interactive fixture design

In this methodology, when a design change is transmitted, the system first determines which fixture elements are affected. Checking which element is affected in fact requires an analysis into the various factors such as the accessibility of the face by fixture elements, ability to maintain deterministic positioning and total restraint, and the ability to provide sufficient support. In the present implementation, we only consider the accessibility of the face. This is mainly because, if a fixture element does not access a face anymore due to a modification, none of the other factors will be satisfied. This factor is also readily identified based on the face change information provided by the ARM. In the methodology, if the fixture elements affected are clamps, then only the clamping element selection stage is redone. If the supporting elements are affected, then the supporting elements selection and clamping element selection stages are redone. Finally, if the fixture elements affected are the locators, then all three stages are carried out again. If more than one type of fixture element is affected, then it is taken that the fixture element, which is earlier in the selection sequence is affected.

As an example, it is assumed a change has been made to Product A through the addition of a boss as shown in

Figure 10(a). The resulting change of the intermediate part model is shown in Figure 10(b). When the transmit_design_change method of the process planning ARM is triggered, the ARM identifies that Face 6 of IPM 1 has changed and informs the Fixture Design client. The Fixture Design client then identifies that clamping elements have been affected. It then reloads the new workpiece at the same location and removes the present clamping elements and starts the fixture design process from the clamping stage as shown in Figure 10(c).

A disadvantage of the sequential fixture design process formulation is that it imposes the need to start from a certain stage of the design process. For example, even if the clamps are not affected and the locators are affected, it will be necessary to go through the clamping stage again. The ability to adaptively deal with design changes is therefore limited. In the following section, we reformulate fixture design synthesis using an evolutionary search approach in an aim to provide greater adaptability in dealing with a design change.

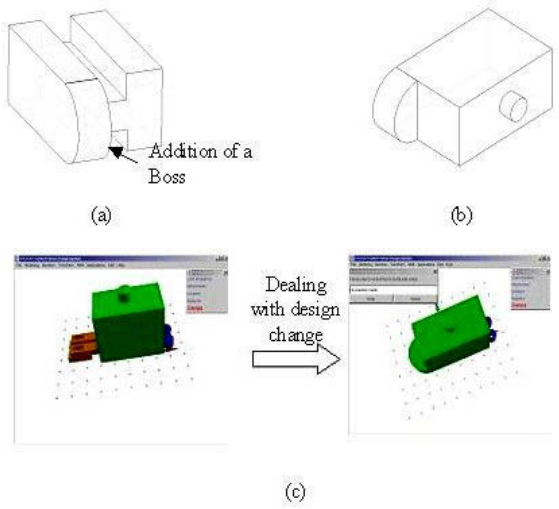

Figure 10(a) Modified part
Figure 10(b) Modified IPM
Figure 10(c) Example of dealing with design change

5.2 Reformulating fixture design synthesis using an evolutionary search approach

In this section, we briefly describe the use of genetic algorithms to synthesize fixtures and the ability to deal with design changes.

In general, a genetic algorithm begins with a randomly generated population of solutions. Each member of the initial population is first evaluated to determine if an optimal or near optimal solution is present in the population. If the termination criterion is met, the GA does not proceed further and the problem is solved. If the termination criterion is not met, a new population is created by the use of genetic operators, reproduction, crossover and mutation. The new population is then evaluated. The procedure is then repeated till the termination criterion is satisfied.

5.2.1 Solution representation

When designing a genetic algorithm, choosing the representation of the solution is a central factor in the success of the algorithm. In our algorithm, the fixture design solution is represented through a tree encoding as shown in Figure 11. In this representation, a fixture solution is divided into locating faces, supporting faces and clamping faces. No restrictions are placed on the number of locating, supporting and clamping faces. Each fixturing face contains fixture elements. Again no restrictions are placed on the number of fixture elements that have contacts with a face. Each fixture element contains attributes element type and element position. Placing no restrictions on the number of fixturing faces and fixture elements allows the solution to adapt according to the conditions of the problem. For example, a workpiece could have a large base and might require a greater number of supports. This representation allows the number of fixture elements on a face to 'grow' to suit the condition. Another example is in the case where a cylindrical hole is used to locate a workpiece. A locating pin would be able to arrest two degrees of freedom and hence, the number of locators required would be less. Thus, the representation can also 'shrink' as required. The generic nature of the representation allows solutions to be sought for any kind of workpiece, without any restrictions.

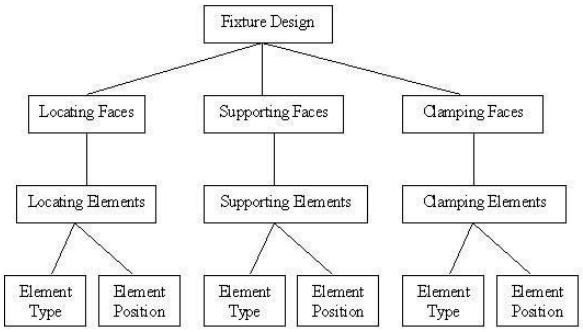

Fig. 11. Representation of fixture design solution

5.2.2 Evaluation of solutions

The evaluation of the solutions is carried out using a simulation approach. The performance of each solution is evaluated by simulating the solution using the interactive fixture design methodology [9]. For each constraint that is violated in the methodology, the fitness

of the solution is reduced. We refer the reader to [9] for the details of the constraints.

5.2.3 Genetic operators

The genetic operators used in the algorithm are reproduction, crossover and mutation. The reproduction operator chooses the individuals in the present population that will create offspring for the next generation. The purpose of selection is to emphasize the fitter individuals in the population in the hope that their offspring will in turn have higher fitness. However, too strong a selection procedure will mean that suboptimal solutions with high fitness values will dominate the population, reducing the diversity needed for evolution. Too weak a solution will result in a slow evolution process. In the present algorithm, a fitness proportionate selection method is utilized.

The reproduction operator selects good solutions to be present in the new generation, but does not create any new solutions. It is the crossover and mutation operators that create new solutions. The crossover operator combines segments of different solutions to create a new solution. An example of a crossover carried out between the two solutions in our algorithm is shown in Figure 12. In this system, the mutation operator is used to create a new solution by changing the fixture element type or position of a randomly selected fixture element.

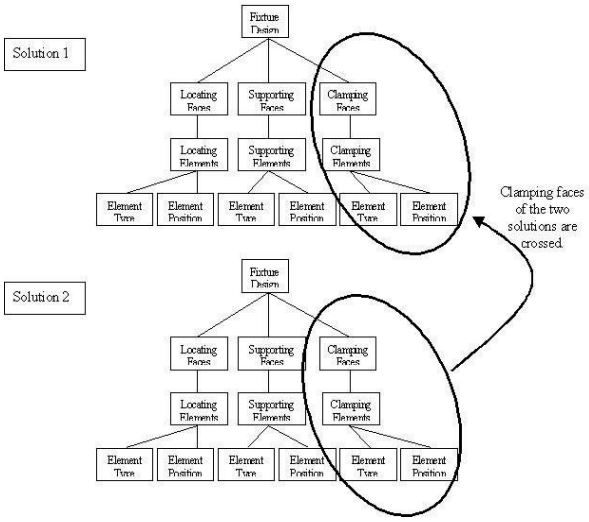

Fig. 12. Example of a crossover

5.2.4 Design changes

An attempt is made in this paper for the GA to adaptively deal with the design changes by developing an appropriate initial population for the GA to evolve. Traditionally, an initial population for a GA is randomly generated. However, in this case, the GA is not attempting to solve an entirely new problem. It is attempting to solve a problem that has been modified from a problem that it had earlier solved. Therefore, an initial population that draws from the previous solutions would provide the GA with a better starting point, leading to a faster route to the final solution. In the present system, when a design change is transmitted, thirty per cent of the final population of solutions for the previous workpiece is copied to the new initial population for the modified workpiece. The thirty per cent of solutions is chosen in a random manner. However, it should be noted that in using previous solutions in the new population, some of the solutions are no longer part of the search space as a result of the design change. For example, a face that could have previously been used as a clamping face could have been deleted. In order to deal with this, before copying previous solutions to the new initial population, the system determines if the solutions are still valid. The ARM provides the information on which faces have been deleted and thus, provides the necessary information to determine if a solution is still valid. If a part of a previous solution is no longer valid, the system randomly generates solutions to make the solution valid. For example, if a previous solution has clamping faces which have been deleted the system generates new random clamping faces and clamping elements. The rest of the initial population consists of entirely new randomly created solutions. The GA process is then carried on as described in earlier.

Combining past solutions with randomly created new solutions allows the GA not only to adaptively deal with design changes but also explore the possibility of arriving at entirely new and improved solutions. Further, in using GA, each solution is evaluated as a whole and there is no need to go through a sequence of activities. Figure 13 shows the graphs of the number of generations against the optimal fitness value for the original workpiece and the modified workpiece. From the figure it is clearly evident that the number of generations for the modified workpiece is far less than the original workpiece, suggesting the ability of the GA to adaptively deal with the design change.

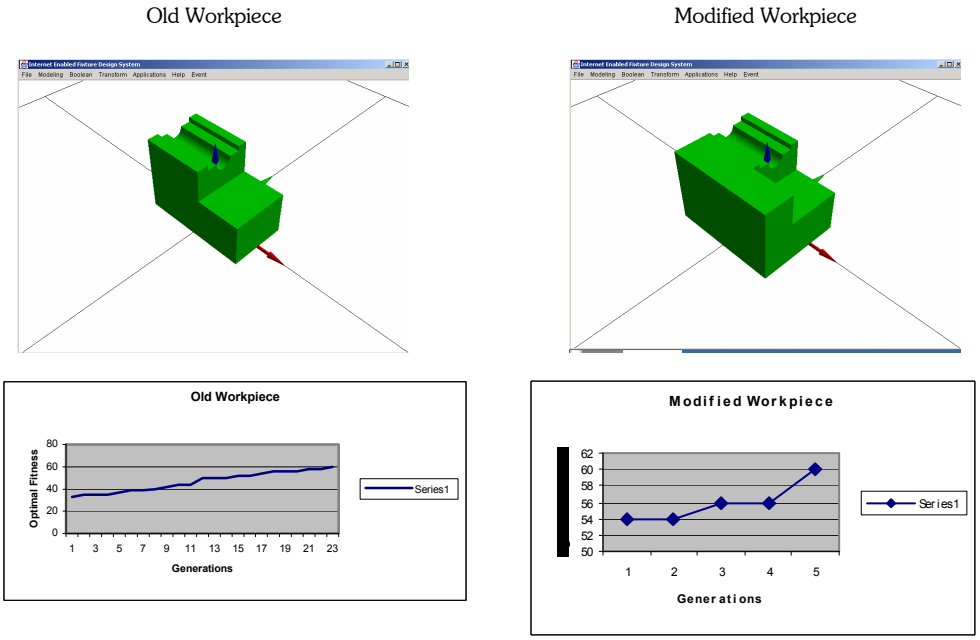

Fig. 13. Fitness vs. generations graphs for old and modified workpieces

6. CONCLUSIONS

This paper has presented the use of a common manufacturing application middleware service known as the Application Relationship Manager as a means to propagate design changes to all related product and process design applications. Downstream applications create relationships with the geometric elements of a product. Geometric elements provide a generic, but meaningful means to create relationships. Deploying the ARM on different applications creates a hierarchy of relationships. The hierarchy creates a clean separation between applications that have direct relationships and indirect relationships.

Dealing with design changes is domain specific and depends on the formulation of the domain process. Using fixture design as an example, we formulated the fixture design domain process using two different approaches. The sequential fixture design process formulation imposes the need to start from a certain stage of the design process, thereby limiting the ability to adaptively deal with design changes. The GA's ability to stochastically alter candidate solutions according to the performance of the solutions provides a more effective way to adaptively deal with design changes.

7. REFERENCES

[1] *Bidarra, R., van den Berg, E. and Bronsvoort, W.F, Collaborative Modeling with Features,* In: CD-ROM Proceedings of the 2001 ASME Computers and Information in Engineering Conference, 9-12 September, Pittsburgh, PA, ASME, NY

[2] Cutkosky M.R, Engelmore R S, Fikes R E, Genesereth M R, Gruber T R, Mark W S, Tenenbaum J M and Weber J C, PACT: an experiment in integrating concurrent engineering systems**,** Computer, 1993, Vol. 26, pp28-37.

[3] de Kraker K J, Dohmen M and Bronsvoort W F, Maintaining multiple views in feature modelling, 4th Symposium on Solid Modeling and Applications, ACM Press, 1997, pp.123-130.

[4] De Martino T, Falcidieno B and Haszinger S, Design and engineering process integration through multiple view intermediate modeler in a distributed object oriented system environment, Computer Aided Design, 1998:30(6), pp. 437-452.

[5] Hoffman C M and Joan-Arinyo R, CAD and the product master model, Computer Aided Design, 1998:30, pp. 905-919.

[6] Jha K and Gurumoorthy B, Automatic propagation of feature modifications across domains, Computer Aided Design, 2000:32(12) pp. 691-706

[7] Jha K and Gurumoorthy B, Multiple feature interpretation across domains, Computers in Industry, 2000:42(1), pp. 13-32.

[8] 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) pp 679-689

[9] 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.

[10] Research Technology Institute, Interoperability Cost Analysis of the U.S. Automotive Supply Chain. (Gaithersburg, MD: NIST Planning Report 1999).

[11] Richard E. Schantz and Douglas C. Schmidt, Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications, *Encyclopedia of Software Engineering*, 2001, edited by John Marciniak and George Telecki, Wiley and Sons.

[12] U.S Department of Defense, Guide to Integrated Product and Process Development, http://www.acq.osd.mil/io/se/ippd/

[13] Wang F-C and Wright P K, "Web-based Design Tools for a Networked Manufacturing Service", 1998 ASME Design Technical Conference, Atlanta, GA, Sept. 13-16.