COMPUTER-AIDED DESIGN & APPLICATIONS, 2018
VOL. 15, NO. 1, 34–46
https://doi.org/10.1080/16864360.2017.1353736

Hybrid state transactional database for product lifecycle management features in a multi-engineer synchronous heterogeneous CAD environment

Devin Shumway, Jonathan Sadler and John L. Salmon

Brigham Young University, USA

ABSTRACT
As interoperability between Computer Aided Design (CAD) systems becomes a possibility, a need arises for a way for the Neutral Parametric Canonical Form (NPCF), as designed at the BYU Site of the NSF Center for e-Design, to be integrated with Product Lifecycle Management (PLM). The only method currently available to users to sync with a PLM system at this time would be to choose one CAD system and create files based off of the NPCF data then save those part files in the PLM system. This database expansion to the NPCF allows the NPCF to hold the entire part history as well as enable future work revision history and configuration management. Enforcing referential integrity within the database allows for part data to never get corrupted and the NPCF allows any CAD system with the appropriate plug-ins to read the uncorrupted data.

KEYWORDS
Interoperability; CAD Database; PLM; Multi-User

1. Introduction

Due to the number of Computer-Aided Design (CAD) applications that are used in a manufacturing supply chain, companies are forced to interface with differing CAD file formats than the CAD system that is used in house. Currently, that is done using translation practices, namely the International Graphics Exchange Standard (IGES) [19] or the Standard for the Exchange of Product Model Data (STEP) [16]. These standards have their benefits, but each has their limitations as well [1, 14]. IGES represents only geometric data contained within the model. While IGES improves upon this limitation, only Boundary Representation (BREP) data is translated and features such as associativity are lost. With STEP, current research is being done on Solid Model Construction History (SMCH) which stores BREP data [3] and construction history [4] to not only keep the geometry but tries to preserve design intent [7]. This leads to large file sizes [17] and results in failure rates as great as 50% [6]. The objective of this research is to reduce file sizes further by reducing repeated data as well as reduce failure rates by denying corrupted data from being translated in the first place.

1.1. Neutral parametric canonical form

PLM refers to any system or software that tries to manage data relevant to a certain product [20]. For the purposes of this research, PLM will specifically refer to the systems that attempt to manage CAD system data. The neutral parametric canonical form (NPCF) is a new neutral format for storing CAD information in a base mathematical definition [5]. In each CAD system a plugin is written that takes every feature that has been created and using that CAD systems application programming interface (API) breaks the feature down into its mathematical definition. This mathematical definition for a feature is the NPCF. The NPCF has been implemented into a database format or the neutral parametric database (NPDB). Due to the NPCF being a neutral format with information to be accessed by many different CAD systems, no CAD-based PLM system exists to store NPCF information for use in Product Lifecycle Management (PLM). The data can be stored in PLM systems but many PLM features such as configuration management would not be available. Typical interaction between the CAD system and the PLM system requires the storage of binary part files such as NX's.prt files, or CATIA's.catprt files. There is no way currently to store PLM data in a synchronous, collaborative, multi-engineer, interoperable database environment. To save a part into a PLM database currently while using the NPCF and NPDB, the user must convert to the system that is interacting with their PLM service and save the file directly. Due to this conversion between systems, the benefits gained from having a multi-engineer synchronous heterogeneous database are lost because the

CONTACT Devin Shumway devin.shumway@gmail.com; Jonathan Sadler jonathansdlr@gmail.com; John L. Salmon johnsalmon@byu.edu

saving process defaults to single user and single CAD system reliant. Another issue is that currently saved CAD files are typically stored in binary and lack the ability to retain referential integrity, the parts files can become corrupted. Within a database format, the product exists as a mathematical representation of the part and is inherently stable.

The NPCF format developed at BYU helps to solve some of the translational problems between current CAD systems by providing full geometric data through the state-based format in which the data is stored [5]. This data is feature specific and thus maintains design intent. However, unlike STEP, the database format developed at BYU for the storage of NPCF data does not allow for the storage of the construction history, which does result in a lack of full design intent being saved. The database of the NPCF stores the feature data for every feature present in a CAD file, part, or assembly. These features are stored in the database and every client with access to the database can access the data that relies on that client's CAD system to compute and display the data through exposing the particular system's application programming interface (API). This allows any user to not only access the data in the database but to push data into the database and receive data when it is uploaded. The server monitors traffic and simply pushes a message to other clients who then process the data locally, resulting in a thick client, thin server architecture [18]. This architecture supports a multi-user synchronous collaborative environment where multiple users can be modeling in the same part and or assembly. This environment is also heterogeneous which allows one user to be working in Siemens NX and another user to be working on Dassault's CATIA simultaneously in the same part or assembly receiving live updates from the other clients.

While the NPCF helps to solve many issues surrounding CAD interoperability, it does leave the user with a deflated user interface because the NPCF has not been developed for all features yet, in which they lose many of the actions that they could perform in a typical CAD system. Some of these features include copying parts, undo, redo, returning to a previous position in the feature tree for editing, and returning to a previous work position without deleting. The lack of these features does not represent an impossibility, but rather an avenue of research yet to be explored.

Some of the features listed above are typically handled by PLM systems. This may be an integrated PLM system like Team Center or as simple as a user manually keeping track of all revisions in Excel. Copying parts and revising parts are tracked using integrated PLM systems. In current integrated PLM systems, part files are stored to represent the parts. Since the NPCF uses a database structure to store feature data, instead of copying part files, it is possible to add a reference to the new part giving it a parent part. With a parent part selected, the data in the parent part can be used to recreate the child part on loading, thereby reducing the amount of storage needed for configuration management.

1.2. Multi-engineer synchronous heterogeneous CAD

For example, multiple users can work in multiple systems at the same time and receive updates. In Fig. 1, there are six users working simultaneously on a model of a guided rocket. Two of the users are working in NX, two in CREO and two in CATIA. They are all working on one assembly file and as one user makes a change to any part, that change can be seen on every other screen that has the assembly open. The CATIA user in the bottom right of the screen is working on a servo that moves one of the control flaps. To show additional functionality, this user is working directly in a part file and does not have the assembly part open. As this user makes changes to the servo, every user that has the assembly file open still receives their change to the servo but the user does not receive any updates to the assembly because they do not have it open. The plugins to the different CAD systems used in this example utilize the NPCF and the NPDB and make up the program called Interop.

Interop, a program created at BYU in the CAD lab, is a thin server, thick client synchronous heterogeneous CAD modeling program. This program uses the NPCF to formulate its database and allow users to work in multiple CAD systems simultaneously [5] similar to the idea postulated for neutral modeling commands by Li [10, 11, 12]. The current database format has tables for each supported feature. These tables have a parent part and when the part is loaded into any system for which a plugin to the Interop environment has been written, that system takes the data from the features that have the current part as their parent part and creates them locally. When a feature is edited, the data for that feature is overwritten so that there is only one record of that feature in the database at any time. Therefore, there is no traditional interpretation of undo, redo, or history for the part. If the feature is deleted it is kept in the database but a delete flag is inserted so that the system knows the feature has been deleted and should be removed as soon as possible on the other clients. This is similar to current CAD system files in that the database contains the information for a state and this state and associated tables are equivalent to the save file for the part.

This approach to CAD is different in that multiple users can edit the CAD file simultaneously in multiple

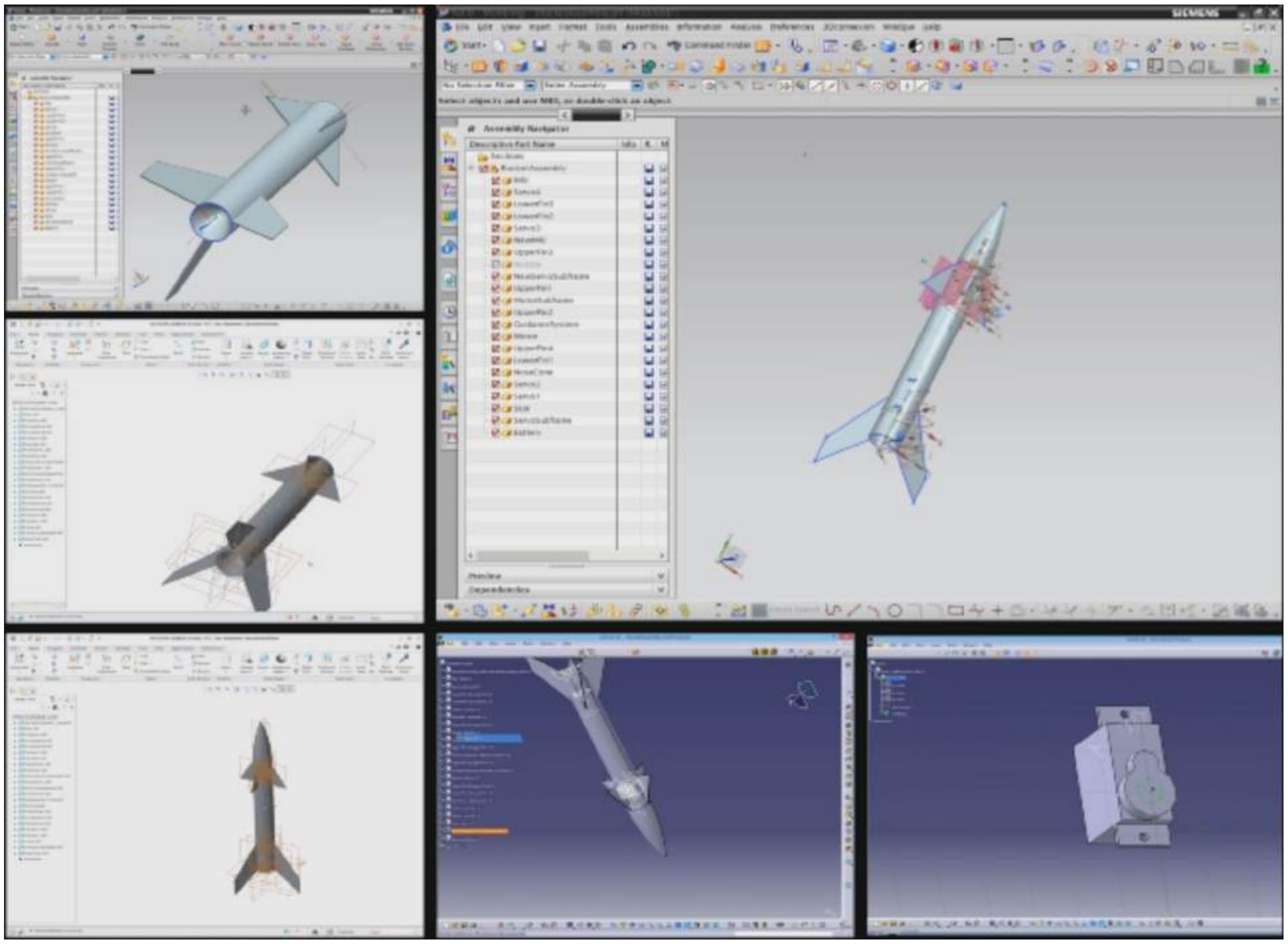


Figure 1. Multiple users working in the same CAD part file in NX pictured in the top left and right corners. CATIA pictured in the bottom right with the blue background and CREO in the two windows in the bottom left.

different CAD systems and receive the edits of all other users in the part in real time. The benefit of this structure, using the NPDB for state saving is quick loading times because the current state of the part is all that is available to be loaded at any time and no previous edits need to be performed to the geometry as was implemented in previous multi-user CAD tools such as NXConnect. The downsides are that with the current system only the current state is saved, no previous data is stored, and only one revision of each part or feature can be saved in the database without completely remaking or copying the part. Design intent is not preserved past the current state of the part but a portion of design intent is preserved because the feature tree is maintained across CAD systems in the current implementation of multi-engineer synchronous heterogeneous (MESH) CAD.

1.3. *Referential integrity*

One of the objectives of this research was to require referential integrity within the NPCF database structure to maintain correct data. Also, included in the objectives was to maintain the quick loading of parts compared to current multi-user CAD packages that rely on loading all features and edits chronologically, while preparing the NPCF database for some PLM capabilities. In the current implementation of the NPCF, quick loading is taken care of through a multiton pattern [2]. The multiton pattern is similar to a singleton pattern except that multiple instances of the class can be created. This enforces all parent features be loaded before a child feature can be instantiated [2]. This speeds up loading times compared to other multi-user CAD systems because the complexity of the loading algorithms is handled within the multiton pattern themselves so that no computations are required for consistency and loading proceeds quickly.

Referential integrity is the process of enforcing every database table to have all foreign keys point to a valid reference [9, 15]. Most database engines will automatically handle references; however, this is only effective when the database has been set up in a way to handle every possible rule. Enforcement of foreign key references in the new NPCF database is vital for keeping the database stable. Referential integrity will help the database recognize corrupted or incomplete data which in turn will be rejected from the database. Rejected data will remain on the client that attempted to send it instead of propagating through each client that is currently in the multi-user session. Referential integrity will also protect the database from corrupt data existing in the part history thus allowing it to be more accurate in storing information. Further checks are implemented on the client side in each CAD system's plug-in to check that data sent to the database is consistent with the feature the user is trying to create.

1.4. Objective

The objective of this research is to define the database structure to allow for full history storage of every NPCF state that the CAD part goes through during the editing process. This data will be the groundwork for integrating PLM features within a synchronous, collaborative, multi-user, interoperable database environment. The research proposed here would not complete a full PLM system, but rather move the NPCF database into a format where the data needed for the PLM features of revision history and configuration management are represented within the NPCF database.

In order for data to be stored for more than just the current state, the NPCF requires a couple of objects attached to the features and for some restructuring of the database itself. The restructure occurred in such a way as to allow a list of the associated feature edits, as well as the feature itself, to be stored. For the state system to remain intact, each feature stored in the database has complete history state data and allows the system to still load quickly without having to apply each edit sequentially. This allows for quick loading of the part no matter what state is loaded, from the first sketch or datum placed in the part to the last edit of an extrude. One disadvantage is that this expands the size of the database, although this becomes less of a problem with improved computational speeds and larger memory available. Despite this requirement, data added to the system will still not be duplicated.

2. Methodology

As shown seen in Fig. 2 the conventional method for revision history takes all saved or checked in parts and saves off a binary part file that represents the part at that point. The proposed change to the NPDB will allow for a revision to be tagged after every single change to the part but saves to the part are not necessary and the save button is not implemented with the NPDB but instead saves a

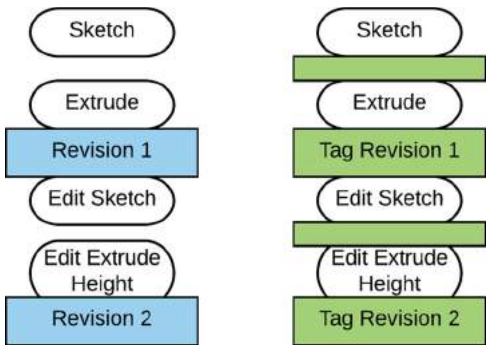


Figure 2. Example showing how conventional revision history works with save points in blue vs the proposed expansion to the NPDB with revisions in green.

normal CAD binary file from the CAD system in which the user is working to their computer. With the part file being set up as a database there is no need to save the part. Instead, each change is added to the database and every single change becomes a revision. To allow for additional control over which revisions are important a user has the ability to tag revisions. This allows for complete control over which revisions are important but no information is lost if revisions are not saved. Tags can further be added at any time.

In order to implement this the NPDB will be revised to include state tables for each feature that store the actual state information for that feature removing it from the feature itself. SQL is used for the database and to implement the code for working with the changes to the NPDB based on the API of each CAD system, for example NXOpen for NX. This approach is a database change that supports maintaining referential integrity and implementing new features into the NPDB for use in MESH CAD.

3. Implementation

The implementation of this database can be broken down into four key operations, which include: revision history, referential integrity, configuration management, and quick loading, and are explained in detail in the following sections.

3.1. Revision history

The main impetus in further developing the NPCF was to include full part history in the database. In order to achieve this, a new database structure was created in the Interop database or neutral parametric database (NPDB) as can be seen in Fig. 3. A DBInteropState table was added that inherits from the main object table DBInteropObject as shown in Fig. 4. This table also points to a DBFeature

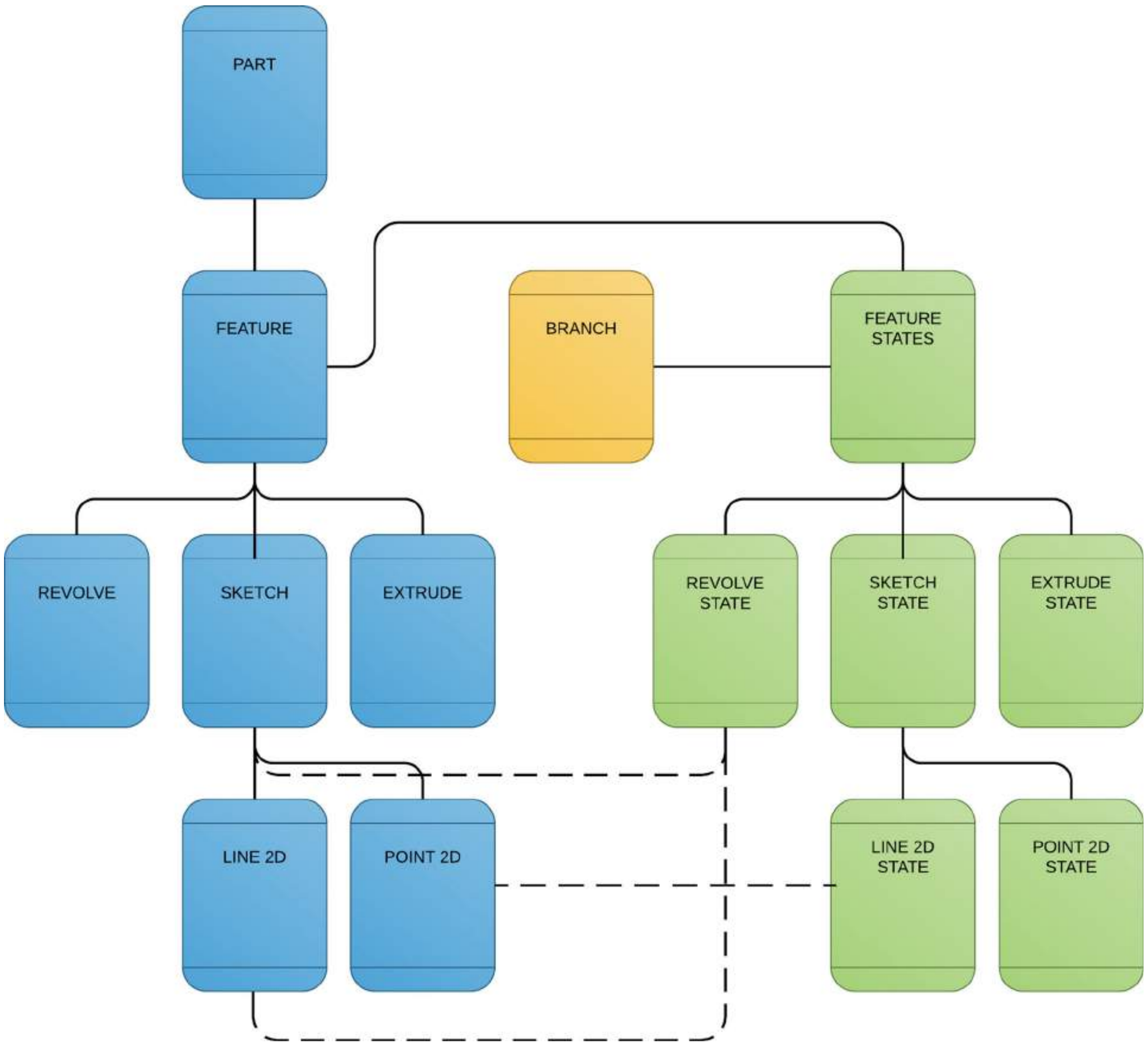


Figure 3. High-level database structure showing added tables for revision history in green and configuration management in yellow. Previous version of database structure can be seen in blue.

table; this relationship is a one to many with one DBFeature having many DBInteropStates. The reason behind this was that each feature has many states and each state is needed to preserve revision history. This small change caused a cascade of other necessary changes to the CAD plugins already included in the Interop environment.

The current state of the NPDB before this change had all database tables as children of the DBFeature table. In order to store all of the feature data in the database, this data was moved under a new table called DBInteropState. The DBFeature tables such as DBExtrude and DBSketch remain as children of DBFeature however they no longer contain the feature data that is stored in the matching DBInteropState table. The DBFeature tables now store a list of the DBInteropStates which then contain all the state information.

When a user creates or edits a feature a new state is created for that object which is a database table entry for a state as can be seen in the right side table of Fig 4. If that state is a new state, a DBFeature is created along with the new state, left side table in Fig. 4. When an edit is made, the DBFeature is first captured from the current state by following the navigational property for the states DBFeature seen in the DBInteropState table in Fig. 4. Then a new state table entry is added to the DBFeature and timestamped accordingly. Thus, if the user creates an extrude object and edits that extrude object multiple times, the database would have one DBFeature table entry for the

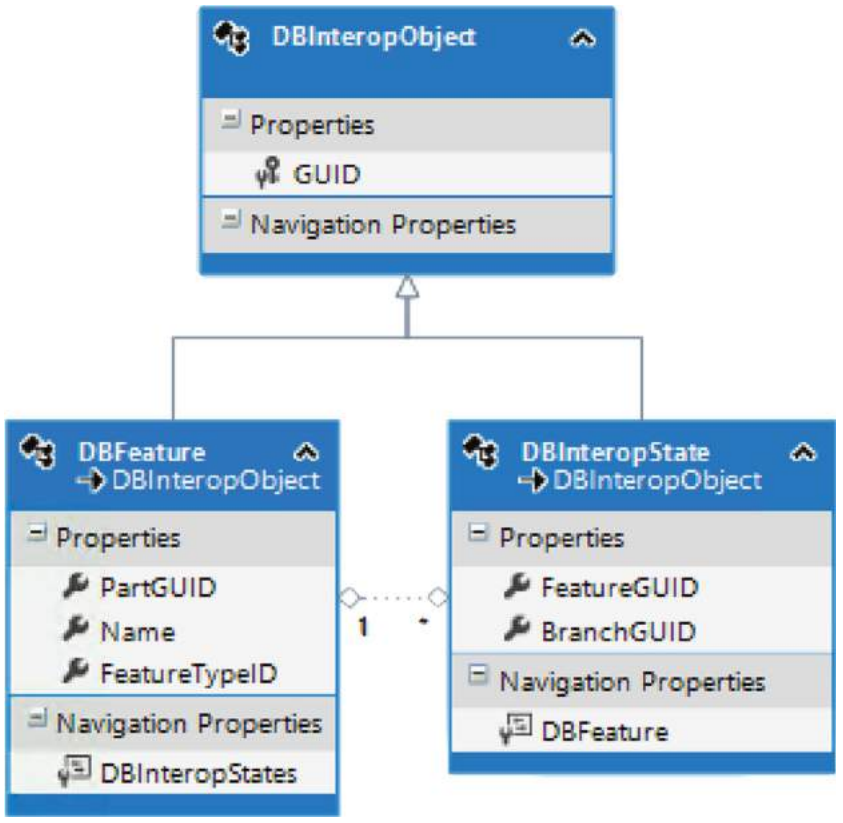


Figure 4. Database tables showing the relationship between DBFeature and DBInteropState with a one to many relationship.

extrude and then several DBInteropState tables that contain the data for the extrude's geometric properties at different times.

3.2. Referential integrity

This implementation creates a database that has an entry for each feature. That database entry has a full list of states for every edit in the part's history. Unfortunately, this method still does not fully address referential integrity as discussed earlier because a DBInteropState can be attached to any DBFeature such as a DBLine2D, which points to the DBFeature table for the end point and the start point. Since DBFeature can be any type of feature this is an issue. However, with the new implementation no other data or tables other than DBFeature is referenced, because all sub tables were deleted. These sub tables were not necessary to set up the references needed to create the states. Now the DBFeature table is all that remains but is not enough to maintain referential integrity. Without referential integrity, data being sent to the server can become corrupted and can be difficult to recover as CAD systems will fail when trying to open the part.

In order to maintain tighter referential integrity, a FeatureTypeID is added to the DBFeature object. The FeatureTypeID is simply an integer value that corresponds to a list and every feature is assigned an integer value. This FeatureTypeID acts as a check for when a client adds a state to a feature. If the FeatureTypeID matches the ID of the state object then the addition is allowed. If the FeatureTypeID does not match then the addition is not allowed and the addition is rejected from the database as it implies that the state was not created in the correct manner. This change to the database structure helps maintain tighter referential integrity.

DBInteropState is the parent table of many different state tables including a number of sketch tables such as DBPoint2DState and DBLine2DState. These tables store the state data for the points and lines that reside within a sketch. Inside the DBLine2DState, there are two navigation properties that are the links that show which points in the database are the associated start and end points for that line as can be seen in Fig. 5. With the addition of states to the database, the endpoint and the start point can each contain a list of states. These states are contained within a DBFeature's DBInteropStates reference list. Thus, a line state has a reference to a start point, which has a list of states for every revision of that point. The challenge is that with the change in the database every reference goes back to a DBFeature object because that is what contains the state list. Although the state list

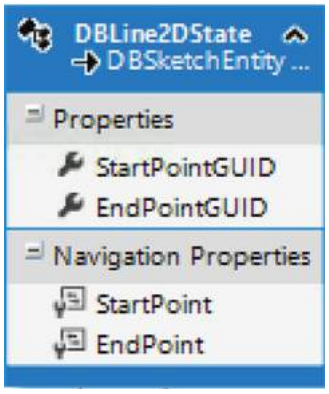

DBLine2DState
DBSketchEntity ...
Properties
StartPointGUID
EndPointGUID
Navigation Properties
StartPoint
EndPoint

Figure 5. Line2DState table showing the references to Start Point and End Point.

does not allow for different states to be stored, from the perspective of the DBLine2DState table there is no way to know if the reference to its start point is actually a point2D or an extrude. Therefore, this method does not maintain referential integrity.

In order to maintain referential integrity, a matching schema under DBFeature was developed that mimics the architecture under DBInteropState. For every state table, a matching DBFeature object table such as DBLine2D that inherits from DBFeature was added. This way, when a feature such as DBLine2DState references two external DBFeatures referential integrity can be enforced through the existence of DBPoint2D as a table. The DBLine2DState has two references in it that are one-to-one; they each point to a separate DBPoint2D object and each of these points has their own state tables. The correct state is chosen to represent that line at any given time. This table is nearly blank, containing an inherited global unique identifier (GUID) from the parent DBFeature that allows the table to be referenced. This forces an external reference that is contained in a state table to point to a DBFeature that is an instantiation of the feature that is actually desired, thus maintaining referential integrity.

This structure also allows the database to store a DBLine2D state that never needs to change when the referenced points are edited. It does this by allowing the points to have separate DBPoint2DState tables which can have their own states. This way, when an edit is made to a point, a state table is added to that point feature, but it is not needed to change the DBLine2D or add a state to it because it has been handled by the point state table and no duplication of data is necessary. For example, if the user has a line that has start point A and end point B, and the user edits start point A to a new location, a new point state is created. However, no new state is needed for the line because it did not receive a new start point table but rather the existing start point was edited.

3.3. Configuration management

Configuration management is the process of producing and managing different configurations of a part that all contain geometry from other configurations. An example of this would be a car manufacturer having two CAD assembly files for their regular and luxury models of each of their designs. These two files expectedly share a lot of the same geometry. For example, they may have the same seat belts designs and they may have the same audio system and all its associated parts. This type of data is all relevant for making the various models of the car. If a change is made in a part to a feature, then currently a new part file with a revision number at the end is created. Continuing with the car example, if a car company desired their sport edition model to have a hood scoop, they would keep the geometry of the rest of the hood and add a scoop to the center. This results in a new part file and this new part file duplicates a lot of the data that is contained in the original part file. To implement a new database structure that would allow the user to add a hood scoop to one part file and keep the original part file intact for their other model, a method from computer programming was applied called “branching” [13].

In order to keep track of branching, a branchID was added in the form of a GUID to the DBInteropState table which can be seen in Fig. 4. A user can create a new branch manually which creates a new branchID. This branchID serves as a unique identifier for the current branch that the state is created on. Figure 6 shows three points being created and edited across multiple branches. This example shows how multiple users can take advantage of the branching functionality and the new approach to configuration management where configurations are handled at a feature level and not a part level.

Figure 6 shows a part that has four different desired configurations labeled Branches 1 through 4. In Branch 1 multiple users start out in the part. Each user can use the CAD system of their choice between NX, CATIA and CREO and one of the users creates the first point in any system while all other users receive the addition of a point in real time. At this point, a couple of users create a new branch which adds a DBBranch table entry for the creation of that branch. They then proceed to create a new point labeled Point 3, edit that point and subsequently edit Point 1. At the same time the users that remain in Branch 1 create a new point labeled Point 2 and edit Points 1 and 2 multiple times. At this point two different users create Branches 3 and 4 and edit Point 1 on each branch. At the end there are four different branches with three points in different configurations, multiple users are allowed in each branch at a time on different CAD systems and when a user makes an edit on a branch other

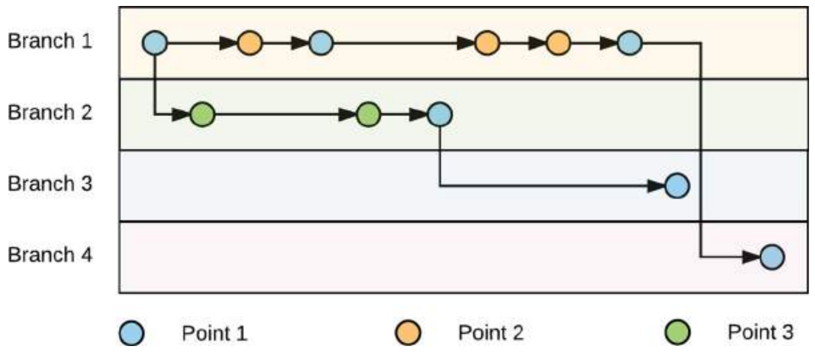


Figure 6. Example timeline of three points being created and edited on different branches at different times.

users do not receive that edit unless they are currently working on the same branch on which the edit was made.

Another example where branching may be useful would be in the example of designing a bike rim as can be seen in Fig. 7. In this example two users enter the part and create the center and outside edge of the rim as can be seen in Fig 7(a). One user creates a second branch of the part and adds a configuration of the part that has a few spokes in Fig. 7(b). At the same time a second user creates a configuration on a separate branch with many spokes as can be seen in Fig. 7(c). In this example both users were able to use the geometry from Fig. 7(a) without saving a part file, copying that part file and uploading the part file into the database. They are also allowed to be working in different CAD systems simultaneously with one users using NX as can be seen in Fig. 6 and the other user using either CATIA or CREO.

This method of configuration management is new by allowing users to create different configurations at the feature level but also new in the fact that users can work across CAD systems simultaneously. A benefit that is added to the design process is that the analytics team could create a branch of the part at any point, take that part and run their analysis while the design team working on the part does not need to check in the part, save or do anything to allow the analytics team to have the most recent version of the part. They can also continue working on the part while the analysis is running, no longer are parts stuck in a silo of design or analysis but can be worked on simultaneously by both parties.

Table 1. Table showing the creation and edits of 3 points on multiple branches in order, according to time with a branch being created in the middle and one point being edited on the new branch.

PointID	XCoord	YCoord	BranchID
1	0	0	1
2	1	1	1
Branch 2	Created	–	–
1	1	4	2
2	1	2	1
3	4	3	1

Table 1 shows a hypothetical, incomplete list of states for a DBPoint2D feature to serve as an example of how this branching method works.

The chain of events as to how this table is created is that a user creates two points: one at coordinates of (0,0) and (1,1), respectively. They then branch off of this part and change point one on the new branch to coordinates (1,4). They then return to branch one and move point two to coordinates (1,2) and create a new third point at coordinates (4,3). (Although this simple scenario demonstrates how this implementation works, the fundamental steps would be identical in the context of a real and more complicated scenario). If two users were actively editing this part concurrently and one was working on branch one and the other was working on branch two, then the user working on branch one at the end of this editing scenario would have three points at coordinates of (0,0) (1,2) and (4,3). The user working on branch two would have two points located at coordinates of (1,4) and (1,2). It is a

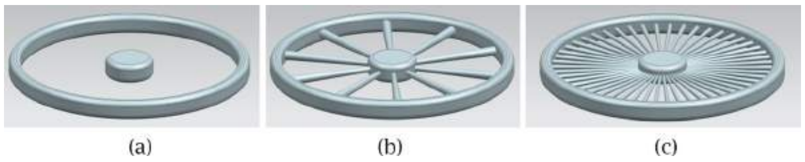


Figure 7. Branching example for a bike rim. The base features being created in (a) and two different configurations of the part being shown in (b) and (c).

simple chronology of events detailing what has happened in the part up to this point in time.

Table 2 shows the branch history of Branch 1 while Tab. 3 shows the history of Branch 2. In order for the full history to be preserved, Branch 2 contains all of the information from Branch 1 as well as all additional changes made after Branch 2 was created. As expected, Branch 1 contains all data for the changes made on Branch 1. Since the point state table really only contains four point states, this schema has successfully created two parts without duplicating data and reducing the storage size of the objects. A downside for this method is that the state based loading scheme, mentioned in the introduction does not hold and loading becomes more complicated. However, data is still not duplicated and one additional step in the loading process is added out from every branch that is created in sequence to overcome this downside.

One other benefit of this process, beyond the scope of this research, is the ability to keep features associative with a parent feature. In one scenario, multiple user could be editing a base feature while working in a different configuration of the part resulting in the part updates based on the changes to the base features. An example of this would be if a car rim designer wanted to create two different rims but both models needed to have the same mating mechanism for connecting to the axle of the car. In this scenario, if a user were to design the mating mechanism which is part of the part and then create a branch where they create the second design of the rim they could have the mechanism be associative to the first part. Therefore, when a change needs to be made to the mating mechanism, it could be made in the first part and automatically be made in the second part. No wave linking would have to take place, no added features would be needed, and the user could select to make that section of the part associative. The potential of this technique would require additional research into design practices and finding use cases where this could be beneficial, but it is an interesting research topic that is now enabled through the full history with associativity and configuration management.

Table 2. Table showing the edits and creations that took place on Branch 1.

PointID	XCoord	YCoord	BranchID
1	0	0	1
2	1	1	1
2	1	2	1
3	4	3	1

Table 3. Table showing the edits and creations that took place to make up Branch 2.

PointID	XCoord	YCoord	BranchID
1	0	0	1
2	1	1	1
1	1	4	2

3.4. State based loading

Previous attempts at multi-user CAD, such as NXConnect, implement a full history transaction database that handles loading by performing all operations performed by users after the last part save in order to create the part [7]. This is a good method of making sure that all users in the part are at the same point and have the same data after loading. The problem is that if the part has not been saved in a long time there are many operations that need to be processed and the part takes a long time to load [8]. Also, within a heterogeneous CAD environment, the CAD files cannot be saved into their respective CAD systems part files without losing the benefits created by having the NPCF, including the fact that every time the part is loaded it would have to be translated back into the NPCF for use by the other systems. A user could have changed their mind multiple times about the height of an extrude, and as a result, the loading process itself would change the extrude multiple times. The benefit of the current state based database is that it stores the current state of the part so only one action is performed per feature and no edits are computed.

Proper application of branching is vital to implementing a fast loading scheme. The current loading scheme can always be fast because it has one state of the part, which is extracted from the database, and then constructs the geometry through the CAD system's API. With the additional information stored in the database, the client needs to know which state is being loaded and on what branch. When the part has multiple branches, the server needs to know which latest version of each feature must be loaded by the client and send that data to the client to open the child branch without errors.

The loading can become fairly complicated when multiple branches exist, but the fundamentals can be described in the terms of the example shown above in Tab. 2 and Tab. 3. In order to load Branch 1, the latest state of each feature with the branchID equal to Branch 1 would be loaded into the part and loading would be complete. Loading for a single branch is as simple as grabbing the last state saved in the database.

For loading Branch 2, the schema loads all of Branch 1 up until the creation of Branch 2. It then loads all updates to those features that exist on Branch 2. The process, however, does not load the updates to those features that exist on Branch 1. Essentially, it loads Branch 1 up until Branch 2 and then switches to Branch 2 and does not execute any of the changes that are on Branch 1. Creating new states is as simple as adding a new state with the appropriate branchID for the current branch.

This approach to loading is not an improvement on traditional CAD part loading for single user systems, in fact since no binary files are used it is slower than traditional loading as the API must perform all of the operations to create the part on load. This approach was designed to allow for loading the most recent state of the part without having to load any previous edits as was available with the previous NPDB due to not having the additional part states saved in the database. To get to the final state of the part as is implemented in NXConnect as mentioned above would have to load every edit. This approach combines the state based loading available with the NPDB in its old form with the full transaction history provided in NXConnect without having to increase loading times, improving on a full transactional database loading time but not on traditional loading times.

4. Results

As a result of this research, the NPDB has been expanded while still utilizing the NPCF to allow for incorporation of revision history and configuration management. These added capabilities were successfully demonstrated by a team of users by modeling each feature supported in the database in NX. Upon successful creation of each feature, users were asked to edit each feature. After successful edits were made to each feature and propagated across all clients, the database was manually checked to ensure that data was stored correctly. No specific model was used and users had the freedom to create features in whichever manner they saw fit. Edits were done in the same fashion allowing users to make changes as needed. This validated that NPDB with the end model containing every feature supported in the NPDB at this time.

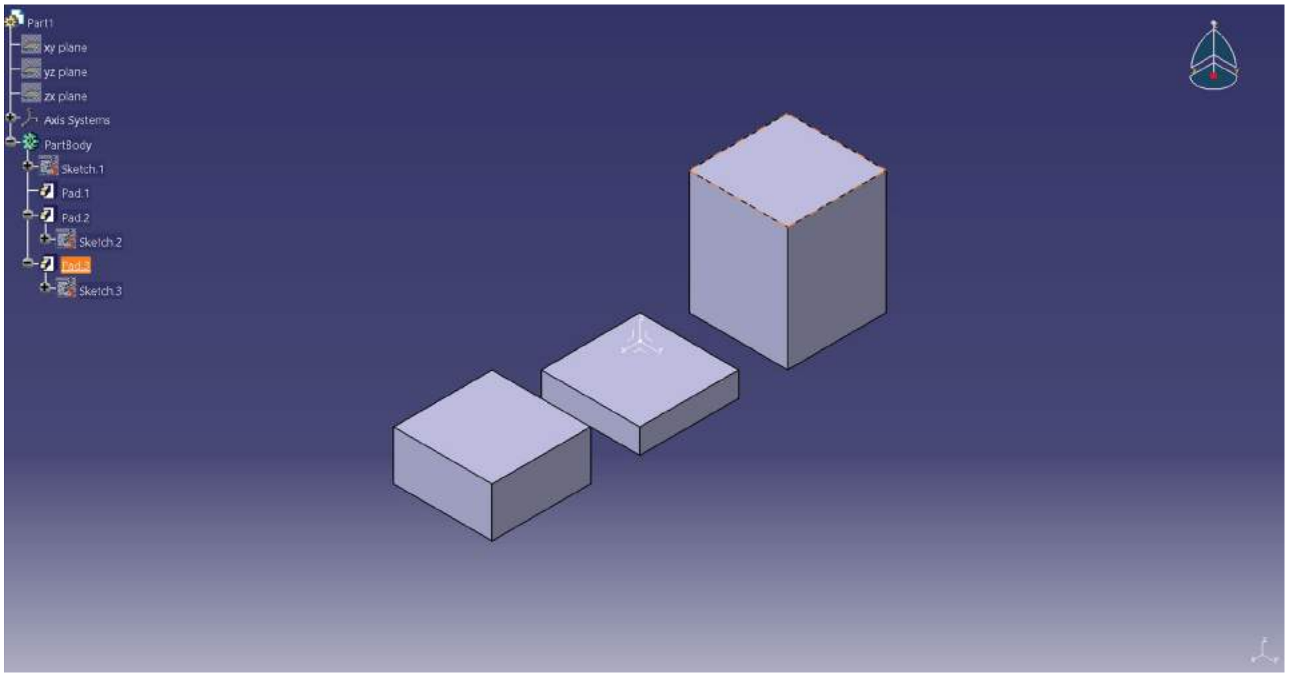
In order to test the NPDB expansion across multiple CAD systems, the extrude feature was implemented in the plugins for CATIA, CREO and NX. Three users each working in a different CAD system at the same time on different machines modeled a simple extrude as seen in Fig. 8 (a) and edited the height of the extrude multiple times as can also be seen for comparison in Fig. 8 (a). Each client was able to pull the extrude made by the other clients and apply them on their own machine as well as update the model and receive edits on their machine. As before, in order to ensure that data was pushed for not only the creation of the extrude but also for each subsequent edit the database was checked manually and verified its implementation as can be seen in Fig 8 (b). This validates the process because the data shows the different limits used by the multiple clients on the three different Referenced Profile GUIDs. Each state contains its identifying GUID, while each DBFeature is associated with multiple states demonstrating proper function of the database where each state should have their own GUID which is represented in Fig. 8 (b) in the left column.

To test for the correct implementation of referential integrity, users were asked to constrain a sketch line to a point that was outside of the current sketch in three-dimensional space which should have a table identity of Point3D. Line2D has references for start and end points that only allow Point2D as seen in Fig. 5. Since the point being referenced was a Point3D but the database expected a Point 2D, this data was rejected from the database and the other users in the part did not receive this edit. In a full CAD system, this action would be allowed, so further research is required to allow for multiple different features of objects to be used as reference objects.

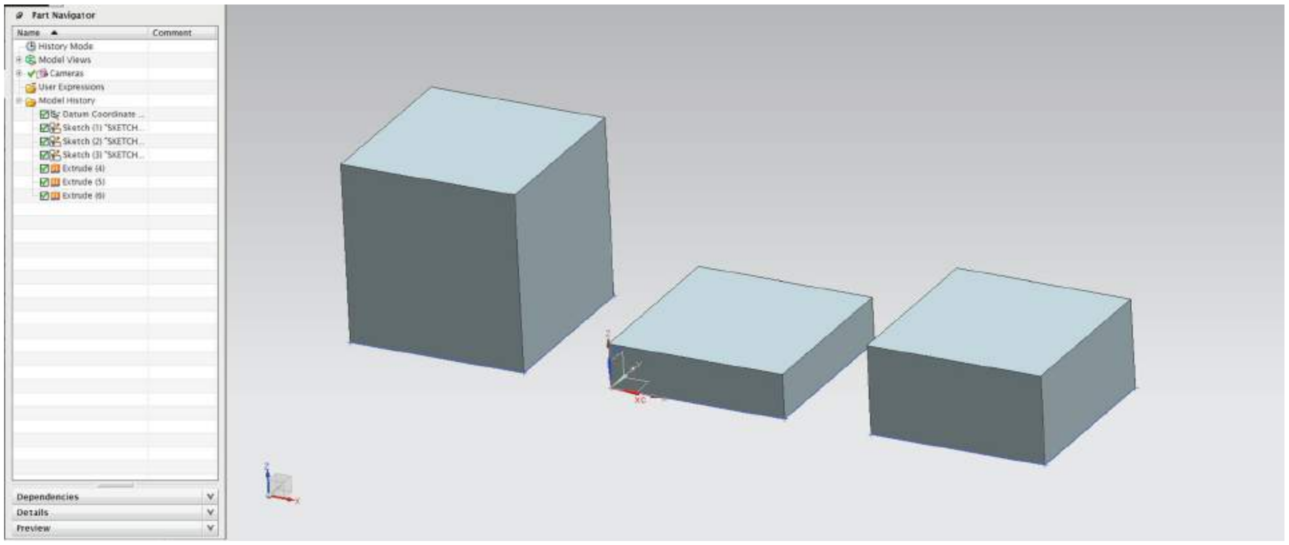
Branching separate parts is functional in the database and the database knows how to handle the received information. However, no method has been instantiated in the CAD plug-ins at the moment that would allow the user to access this functionality. For testing purposes, the branchIDs were entered manually to test the functionality of the database on receiving the data. The database worked as expected in this regard rejecting bad data and storing the data correctly for multiple different branches.

Loading is fast and allows users to have created multiple different branches in the database but shows all branches as a single part. Although this may not be desired in the future, the proof of concept database as part of this research that would handle the multiplicity of branches was demonstrated and not necessarily the graphical user interface on how the user would interact with them. For testing these features, the database calls were manually intercepted and a branchID was added to the call so that the database could store the correct data. Database messages were also manually sent with new branch ID's to the other clients to test receiving data that is not in the current branch. Further research is needed to implement branching of parts locally in the CAD client as well as additional research on how to automate the loading of the data correctly without mending database messages.

The objective of this research was to expand the NPDB to allow for advanced PLM features to be implemented locally and utilize the NPCF. Based on these validation tests, the NPDB has been correctly enhanced and is working appropriately to store all the data for creating, deleting, and editing features in a MESH CAD environment. With further research, additional large add-ons can be achieved to increase the capability of the MESH CAD environment.



(a)



(b)

Figure 8. (a) Final state of extrudes shown in both the CATIA and NX CAD clients. (b) Database table data for extrudes shown in (a) as well as all edit data.

5. Conclusion

5.1. Summary

While MESH CAD and the NPCF have been developed to help increase design transparency across a heterogeneous CAD environment, these solutions are still in the early stages of development and lack full connection into PLM systems which are necessary to the design workflow. In order to better test and develop these systems to help address issues, such as the cost of translation of CAD file data, there is a need to allow interaction with features that users are accustomed to having in a PLM system. In a multi-user environment, it becomes increasingly important to know what other designers were intending to do while creating a part and allowing a user to see each change made to the part in correct chronological order thereby moving toward greater visibility between designers. Keeping the database in a state in which referential integrity is maintained helps keep the data passed between designers valid and allows for greater trust in models that are being developed by multiple designers at the same time. Configuration management tools allow users to select features needed for a new design without having to copy data or duplicate work, which allows for quicker modeling practices and a faster turnaround time during the design process. These features add to the validity of the multi-user environment and the NPCF.

PLM systems manage great amounts of data and the changes made to the NPDB only account for a very select feature set managed by PLM. These features however increase the capabilities available to a multi-user environment and the small feature set allows for them to be implemented securely and accurately. Future work will allow for greater control over CAD data in any system and a greater decrease for the time and money required currently for translation.

5.2. Future work

As mentioned, currently in order to create a branch the database tables must be edited manually. Ultimately, a GUI could be implemented to create a new branch or configuration of a part. This feature would expand the functionality of the MESH CAD environment allowing for configuration management. Due to the nature of the NPDB, this configuration management could be enabled at the feature level. This is different than the current state of configuration management which is only done at the part level. Additional research would be required to understand if this additional functionality is beneficial to the design process. In order to facilitate this change, multiple changes are needed to each of the MESH CAD plug-ins, including the creation of a GUI with which to interact to create a new branch. This will allow for quick adjustments to be made to the configuration of assemblies, subassemblies, and features. An additional feature that would be beneficial to a user would be the ability to create a new configuration based off a previous revision of the part. In order to facilitate this change a different GUI would be required to allow the user to revert to a previous state of the part and then create a branch from that point. These features should allow for the addition of the functionality that is desired by the expansion to the NPDB and, having these features available in the CAD systems, could result in a greater design transferability and transparency. It could also lead to a reduction in time required to make different configurations of the parts which decreases the time from conception to part realization.

These changes are important because they help to integrate PLM features into the CAD system giving more power to the designers and allowing for more collaboration. With increased collaboration, it is believed that there will be fewer turn backs in the design process due to errors and more innovation due to the ability to collaborate with other users in real time.

Acknowledgments

This research was in part supported through membership funding through the National Science Foundation Center for e-Design – Brigham Young University Site.

ORCID

Devin Shumway http://orcid.org/0000-0002-5186-7857
John L. Salmon http://orcid.org/0000-0002-8073-3655

References

[1] Basu, D.; Kumar, S. S.: Importing mesh entities through IGES/PDES, *Advances in Engineering Software*, 23(3), 1995, 151–161. http://doi.org/10.1016/0965-9978(95)00075-5

[2] Bowman, K.; Shumway D.; Jensen G.: Pseudo-Singleton Pattern and Agnostic Business Layer for Multi-Engineer, Synchronous, Heterogeneous CAD. In press http://doi.org/10.1080/16864360.2016.1240449.

[3] Cheng, Y.; He F.; Wu, Y.; Zhang, D.: Meta-operation Conflict Resolution for Human-Human Interaction in Collaborative Feature-Based CAD Systems, *Cluster Computing*, 19(1), 2016, 237–253. http://doi.org/10.1007/s10586-016-0538-0.

[4] Choi, G. H.; Mun, D.; Han, S.: Exchange of CAD part models based on the macro-parametric approach, *International Journal of CAD/CAM*, 2(1), 2009.

[5] Freeman, R. S.; Bowman, K. E.; Red, E.; Staves, D. R.: Neutral Parametric Canonical Form for 2D and 3D Wireframe CAD Geometry. In *ASME 2015 International Mechanical Engineering Congress and Exposition*, 2015, V011T14A004-V011T14A004, American Society of Mechanical Engineers. http://doi.org/10.1115/IMECE2015-51969

[6] Haenisch, J.: (1990, November). CAD-exchange-towards a first step implementation. In Industrial Electronics Society, *IECON'90, 16th Annual Conference of IEEE*, 1990, 734–739, IEEE. http://doi.org/10.1109/IECON.1990.149231

[7] Han, S.: Macro-parametric: an approach for the history-based parametrics, in Soonhung Han (guest editor), Special issue: The future of CAD interoperability: History-based parametrics, *Int. J. Product Lifecycle Management (IJPLM)*, 4(4), 2010, 321–325. http://doi.org/10.1504/IJPLM.2010.036485

[8] Hepworth, A. I.; Tew, K.; Nysetvold, T.; Bennett, M.; Jensen, G.: Automated Conflict Avoidance in Multi-user CAD, *Computer-Aided Design and Applications*, 11(2), 2014, 141–152. http://doi.org/ 10.1080/16864360.2014.846070.

[9] Kappe, F.: (1996). A scalable architecture for maintaining referential integrity in distributed information systems. In J. UCS *The Journal of Universal Computer Science*, 1996, 84–104, Springer Berlin Heidelberg. http://doi.org/10.1007/978-3-642-80350-5_8

[10] Li, M.; Gao, S.; Wang, C. C. L.: Real-Time Collaborative Design With Heterogeneous CAD Systems Based on Neutral Modeling Commands, *Journal of Computing and Information Science in Engineering*, 7(2), 2007, 12–15. http://doi.org/10.1115/1.2720880.

[11] Li, M.; Gau, S.; Li, J.; Yang, Y.: An approach to supporting Synchronized Collaborative Design within Heterogeneous CAD Systems, *ASME 2004 International Design Engineering Technical Conferences*, 2004, 511–519. http://doi.org/10.1115/DETC2004-57703.

[12] Li, M.; Yang Y.; Li, J.; Gao, S.: A preliminary study on synchronized collaborative design based on heterogeneous CAD systems, PhD thesis, Zhejiang University, 2003. http://doi.org/10.1109/CACWD.2004.1349025.

[13] Loeliger, J.; McCullough, M.: Version Control with Git: Powerful tools and techniques for collaborative software development. *"O'Reilly Media, Inc."*, 2012.

[14] Marjudi, S.; Amran M.; Abdullah, K. A.; Widyarto, S.; Majid, N.; Sulaiman, R.: A Review and Comparison of IGES and STEP, *Proceedings of World Academy of Science, Engineering And Technology*. 62, 2010, 1013–1017.

[15] Markowitz, V. M.: Safe referential integrity structures in relational databases (No. LBL-28363; CONF-9109190–1). Lawrence Berkeley Lab., CA (USA), 1991.

[16] Pratt, M. J.: Introduction to ISO 10303—the STEP standard for product data exchange, *Journal of Computing and Information Science in Engineering*, 1(1), 2001, 102–103. http://doi.org/10.1115/1.1354995

[17] Pratt, M.J.: Extension of the Standard ISO10303 (STEP) for the exchange of parametric and variational CAD Models, *Proceedings of the Tenth International IFIP WG*, 5(3), 1998.

[18] Red, E.; Jensen, G.; Weerakoon, P.; French, D.; Benzley, S.; Merkley, K.: Architectural limitations in multi-user computer-aided engineering applications, *Computer and Information Science*, 6(4), 2013, 1.

[19] Smith, B.; Wellington, J.: Initial graphics exchange specification (IGES); version 3.0 (No. PB-86-199759). US. Nat. Bureau Stand, 1986.

[20] Stark, J.: (2015). Product lifecycle management. In *Product Lifecycle Management*, 2015, 1-29. Springer International Publishing. http://doi.org/10.1007/978-3-319-17440-2_1