www.cadanda.com
Computer-Aided Design AND APPLICATIONS

Scalable Integration of Commercial File Types in Multi-User CAD

Ammon I. Hepworth[1], Thomas Nysetvold[2], Joshua Bennett[3], Glen Phelps[4] and C. Greg Jensen[5]

[1]Brigham Young University, ammon.hepworth@gmail.com
[2]Brigham Young University, tom.nysetvold@gmail.com
[3]Brigham Young University, joshjb17@gmail.com
[4]Brigham Young University, wilysword@yahoo.com
[5]Brigham Young University, cjensen@byu.edu

ABSTRACT

Current commercial computer aided design (CAD) tools limit a parallel engineering design workflow by only allowing a single user in the CAD model at a time. The NSF Center for e-Design at BYU has recently developed multi-user CAD tools which enable a parallel design workflow by allowing multiple users to simultaneously contribute to the same CAD model in real time. The combined challenges of consistent distributed naming and robust interoperation with commercial file types have created scalability and usability issues for previous multi-user CAD implementations. This paper presents persistent naming methods and a file-based architecture that address these challenges. An implementation of these methods shows that multi-user design within commercial CAD is increasingly scalable.

Keywords: collaborative design, concurrent engineering, multi-user CAD, CAE.

1. INTRODUCTION

Current commercial computer aided design (CAD) tools limit concurrent engineering, or the ability of product development teams to work in parallel, by only allowing a single user in the CAD environment at a time [16]. The NSF Center for e-Design, Brigham Young University (BYU) has developed multi-user CAD tools which enhance concurrent engineering capability by allowing multiple users to simultaneously contribute to the same part or assembly in real time. This includes the ability to view, add and modify the same part concurrently in the same environment. As each user adds to, removes from and/or edits a part, each user visualizes contributions made by all other collaborators to that part in real time. The ability to allow multiple users to simultaneously contribute to the creation of the part truly enables a parallel work environment in the CAD system [16].

The implementation of such a multi-user collaborative environment requires a system architecture to support data transfer between various users. The system implemented by BYU utilizes a strong client and thin server in a client-server architecture. Each client runs a CAD system with a multi-user plugin. As one client performs an operation in the collaborative session, the data for that operation is uploaded to the server, which distributes it to the other clients. The server also maintains a record of all operations performed by each user. When the other clients receive the operation data, functions are automatically called to perform the operation on that client's session. In this way, all clients have a representation of the same shared CAD model in which multiple users can concurrently contribute. Fig. 1 shows a system diagram of the multi-user CAD architecture.

This architecture has been implemented using the application programming interfaces (API) of commercial CAD systems, including Siemens NX 8.0, Dassault Systemes CATIA and Autodesk Inventor. Doing this has enabled these single-user CAD systems to become concurrent multi-user applications [20].

Extending a single-user CAD system to be multi-user requires the definition of a format for data exchange, so that the most up-to-date state of the model can be stored on the server and so that operations performed on one user can be transmitted to other users. The server uses a database to store this data, as shown in Fig. 1. Each type of operation requires its own explicitly defined database representation and so feature types must be made multi-user

Taylor & Francis
Taylor & Francis Group

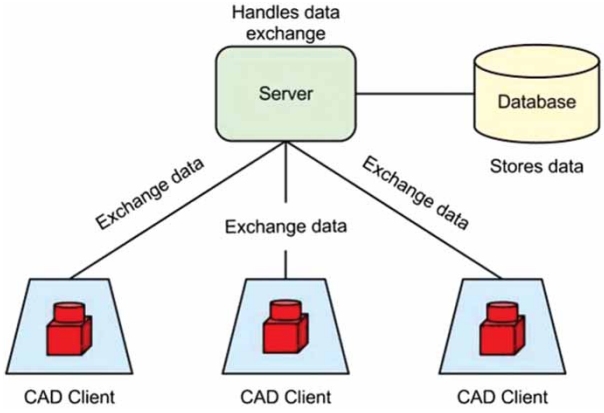

Fig. 1: Multi-user CAD architecture.

one by one. In practice, given the huge and growing variety of data types within any given CAD system, explicitly creating database representations for all of them is very time consuming. Therefore, attempting translation has almost always resulted in some degree of data loss. Because of these and other challenges, no prior system has allowed a real-time, multi-user experience with commercial file types. BYU's early approaches have shown to directly operate with commercial file types with small parts and assemblies. However, it has challenges scaling to large parts and assemblies due to issues with unreasonable load times. This paper presents a scalable methodology and implementation that overcomes this and other related challenges associated with previous multi-user implementations for commercial CAD.

Another issue related to implementing a multi-user environment within an existing commercial single-user CAD tool, as mentioned by Red et al., is that CAD APIs generally do not easily facilitate passing design changes between multiple users. This is illustrated by the fact that when a CAD object is queried on a client's CAD system, it returns a memory address specific to that client, which is not consistent between clients [16]. When an object on one client is changed, the client must be able communicate to the server which object changed so that all other clients can identify, in their own CAD system, which object the change refers to. Each object must therefore be uniquely and consistently identified across all clients and the database. As memory address pointers for given objects differ across clients, they cannot be used as a consistent identifier. Object location in the part tree is also not a unique, consistent identifier, because two users could create objects in the same position, and objects may be moved within the part tree. These methods are not guaranteed to be unique, but a system of unique identifiers is essential to a multi-user CAD solution. Methods must exist for finding the unique identifier corresponding to an object in a part and for finding the object corresponding to a unique identifier. This paper discusses a robust method and implementation for supporting topology identification in a multi-user CAD system.

2. BACKGROUND

The background section briefly discusses existing methods in literature related to previous multi-user CAD systems which use non-commercial file types. It also discusses BYU's previous limited success with commercial file type integration. Previous literature pertaining to single and multi-user persistent naming schemes is also discussed, as are its shortcomings in the context of a commercial CAD implementation.

2.1. Commercial File Compatibilities

Various simultaneous multi-user design systems have been developed over the last decade. These systems have primarily relied on a cloud-based data storage system and a central server where all of the modeling operations are performed. Graphical and interaction data is transmitted to and from the clients and server, thus implementing a thin client architecture. These multi-user cloud-based systems include WebSPIFF [3], NetFeature [13], CADDAC [14] and WebCOSMOS [21]. None of these systems integrates with commercial CAD file formats. One system has allowed merging of commercial part files for an asynchronous multi-user experience [17], but its solution does not apply to synchronous multi-user CAD.

Prior BYU systems have also failed to achieve complete interoperability with commercial file formats. NXConnect, a multi-user CAD system currently being developed at the NSF Center for e-Design, BYU, uses the NX 8.0 API and the previously discussed architecture to allow NX to act as a simultaneous multi-user application [4], [11,12], [15,16], [20]. NXConnect has always been able to export models in NX's standard ".prt" format. NXConnnect has also had a limited ability to convert NX part files to its database format. However, the resulting database representation loses all data not yet explicitly supported by NXConnect. BYU's InventorConnect and CATIAConnect multi-user CAD systems have offered the same functionality, with the same challenges. Thus, no prior system we are aware of has allowed synchronous multi-user CAD to completely interoperate with commercial file types.

Since pre-existing corporate data is typically stored in commercial CAD file formats, the ability to interoperate with these file types is a necessity for companies with a large amount of pre-existing CAD data. Furthermore, a server-based multi-user representation may not easily interoperate with pre-existing PLM software (i.e. Siemens Teamcenter, Dassault Systemes ENOVIA). The divide between commercial files and server-based multi-user representations is thus a serious problem for potential users of multi-user CAD.

Server-based file representations can also exhibit scalability problems: loading a model from the server by re-running the commands to generate it is unreasonably slow. For one test assembly including 19 files totaling 43 megabytes, downloading the files from a local server would take seconds and opening the assembly in NX takes approximately 8 seconds. Because running commands programmatically is much more expensive than reloading their recorded results, loading the same model in NXConnect by re-running every command takes over 15 minutes. Larger assemblies would load even more slowly. This loading behavior is very undesirable for practical commercial use; an ideal system for multi-user files would facilitate fast loading of assemblies.

2.2. Single User Persistent Naming

The problem of persistent naming dates back to the original history-based parametric solid modeling CAD systems [8]. The central issue at that time was identifying the topological entities of the solid geometry upon reevaluation from the history tree. Since direct pointers are not persistent upon reevaluation, they could not be used. Simple enumeration methods were also not valid, because model edits may change topology and invalidate the original enumeration [10]. Several authors have presented various solutions to this problem [1,2], [5,7], [10], [19].

2.3. Existing Multi-User Persistent Naming

Jing et al. state that, although in single user CAD systems the topological entities can be named from the modeling history, this does not work in a collaborative multi-user CAD system. This is because operations at various sites may be ordered differently, resulting in an inconsistent modeling history. They present methods to persistently name topological entities including faces, edges and vertices on the level of the geometry kernel. This is accomplished by traversing the topological entities generated by a feature and naming them with a consistent method. They successfully implemented and tested these methods in a prototype collaborative solid modeling system using the ACIS CAD kernel [9].

Although not explicitly mentioned by the author, one main assumption in the method presented by Jing et al. is that topological entities generated by a specific feature are returned in a predictable order. Since some CAD kernels (i.e. Siemens Parasolid kernel) do not return entities in a predictable order, this method can't be applied because it does not incorporate a method to uniquely identify faces, edges and vertices based on criteria other than order [18]. Thus, existing persistent naming methods are inadequate for multi-user CAD implementations using at least one kernel critical to industry today. An identification method based on geometric, topological, and/or feature uniqueness must be employed for robust persistent naming to occur in this situation.

3. PERSISTENT NAMING METHODS

We first present a general method for generating unique identifiers (names) for features and topological entities. A file-linked, multi-user feature naming method is then presented to allow robust, scalable interaction with commercial parametric CAD files. In addition, a unique topology naming method addresses the problem of naming faces, edges and bodies with a CAD kernel that does not return geometry in a predictable order.

3.1. General Naming Methodology

Our method to generate unique identifiers on features and topological entities is as follows:

1. A prefix, '$', identifying the feature as having been named
2. The user's unique username, guaranteeing that each user's features will have names distinct from other users' features
3. The feature type which describes the feature for convenience and readability
4. A sequentially generated integer id, guaranteeing uniqueness from all features on the same client

So, the 57th feature created by a user with username Ivan would receive the name "$Ivan_FeatureType_57," on his client, in the database, and on all other clients. Similar naming conventions apply to all other named entities.

3.2. Hybrid File Multi-User Feature Identification Method

In order to use commercial CAD files with a cloud based multi-user CAD system, a new method has been developed to store database associativity within a CAD file. The method developed takes advantage of user-defined object attributes that are supported in many CAD systems. Instead of creating a system to map memory handles to unique identifiers and storing that map in the client plugin, the unique identifier is instead stored in a user-defined attribute that is directly associated with each object in the CAD part. Storing the unique identifier in the CAD system's native part file makes it possible to restore the state of the multi-user CAD tool, including association data, just by opening a commercial part file. This removes the need to recreate the part object-by-object. This method also applies to assemblies, implementing the same identification method with parts and constraints in the assembly. The new process increases part scalability in multi-user CAD tools

by drastically reducing load times in parts containing several objects. Thus, a part which may have taken several minutes to load using the previous method would only take a few seconds using the new method.

This method provides a solution to many of the challenges previously faced by multi-user CAD systems. Instead of only using a cloud-based file (necessary for multi-user functionality) or a commercial file (necessary for compatibility), the two files are combined in a hybrid. This file is of the same file type as an ordinary part file of the CAD system it was created in, and is fully compatible with the original CAD system with no need for a multi-user plugin.

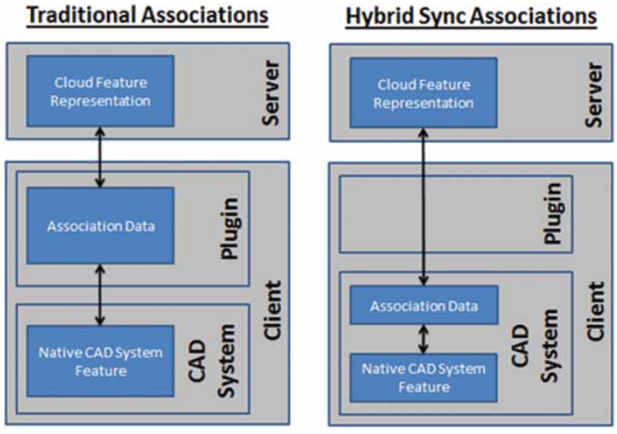

Fig. 2: Hybrid sync associations vs. previous associations.

The new architecture (see Fig. 2) no longer stores data about feature associations (i.e. maps from CAD system object pointers to database identifiers) in the plugin layer. It takes advantage of a capability in many commercial CAD systems which allow users to define their own feature-associated variables within the CAD system, which are then saved in standard CAD files. For example, a user could take a sketch object and attach to it a new variable called "CloudSketchID" that would store the unique identifier of the corresponding sketch object in the cloud. This maintains the association that was formerly managed by the plugin, but in a way that is not affected by closing and reopening the CAD system and/or plugin. In the new architecture, the plugin essentially consists of procedures for relating the CAD data to the cloud data and stores only incidental state data.

Since all data is stored either in the database or in the part file, with none in the multi-user client plugin, this method makes the multi-user client both lighter and more fault-tolerant. If the client crashes, is closed, loses connectivity, etc., no data is lost and simply re-opening the plugin restores its original functioning state with no need to reload the file. Without this method, a time-consuming full file reload is required if the client crashes or becomes out of sync. Therefore this method significantly reduces the reload time for the part to sync with the database.

3.2.1. *Existing Files and File Management*

The method also allows multi-user cloud editing support to be added to pre-existing single user commercial CAD files. This is accomplished through the translation system which identifies all of the features in the file that are compatible with the multi-user plugin, creates database representations for them, and adds a user-defined attribute to each feature associating it with its database analogue. Even features that are not fully supported may at least be identified, with user-defined variables and database analogues. This allows some limited support for features without their own explicit database representations. Features that are not fully multi-user supported will not be editable, but will at least be visible and will never be lost in translation to or from the multi-user file type. Since even non-multi-user supported features are cloud-associated and consistently identified, they can in some cases even be referenced by new multi-user supported features. For example, a sheet body that could not be created or edited in multi-user mode is still able to be used by reference to split a new multi-user solid body.

This method supports a variety of options for file/data management. One option supports having a copy of the CAD file (containing both the file and the database content) automatically saved and maintained on the cloud. Another option supports a seamless integration with commercial PLM/data management systems. In this option, when the part is closed a database dump of the cloud material can be stored in a user variable within the part file, which can then be checked in to the data management system. This is done by either the data management system's normal check-in method for CAD files or by automatically using integration with the data management system's API. This method can also save a simple stand-alone file in the format of the original CAD system, with no cloud or multi-user dependency.

3.2.2. *Offline Editing*

Since the method allows a seamless integration between a part file and multi-user cloud data, the part file can be edited offline by a single user using only the original CAD system (no need for a multi-user client plugin). That user can then perform any action supported by the original CAD system, without regard for the limitation of what is multi-user supported. In addition, he can also edit features that are represented in the database. The next time the part is loaded for a multi-user edit session, it will automatically upload the added multi-user supported features and tag its unsupported features using the same

functionality as previously discussed. Pre-existing features are compared to the original multi-user feature information attached to them to identify differences and then modify the database feature instance if necessary.

Obvious modifications to this same functionality potentially allow multiple users to independently edit a part file offline and then merge their changes; although this approach requires a robust methodology to resolving conflicting edits. The ability to take a part off-line and add or modify non-multi-user supported features facilitates industrial workflows, where a large majority of features are multi-user supported but a few unusual features are occasionally needed.

3.3. Topology Identification Method

The general naming methodology discussed in section 3.1 apply trivially to the identification of features, curves, expressions, dimensions and constraints because they can be named directly when the object is created at the client level. In addition to naming these, the identification of bodies, faces and edges must also be consistent across multiple clients because they are referenced in the creation of features, curves, expressions, dimensions and constraints. However, doing this offers some unique challenges. For example, bodies, edges and faces in the Parasolid geometry kernel are "not returned in any predictable order" [18]. Therefore it requires custom methods to identify the same bodies, faces and edges in a part across clients based on their geometric, topological and feature qualities.

Methods have been developed to identify common bodies, faces and edges across clients and are executed in the following order:

1. Edges are identified by querying data on the edge at discrete values along its length with a given tolerance (see Fig. 3)
   a. Linear edges will only require end point data
   b. Non-linear edges may require more resolution to uniquely identify
2. Faces are identified by querying data at discrete values across the face with given a tolerance (see Fig. 4)
   a. Planar faces will not require discretization if all edges are identified uniquely. This is because they can be inferred from the bounds of the edges
   b. Non-planar faces may require more resolution to uniquely identify
3. Bodies are identified by using all of its faces and edges which are already identified

This approach uniquely identifies bodies, faces and edges except for in two cases: 1) identical topological entities occupy the same space in the part, and 2) all data compared on the entities match, but varies between the discrete samples. The first case is solved by inferring the topological entity from the feature it was generated from to identify its uniqueness. For example, two edges which occupy the same geometric space will always be a product of different feature creation operations. In this way, uniqueness is determined for identical topological entities based on the feature that generated it. The second case is solved by having a sufficiently small enough sampling discretization for edges or faces so as to uniquely identify it. However, a fine discretization resolution may take more computation than is desirable. A method optimized to reduce computation time will provide an iterative approach to refine resolution as required, adding sampling points to compare the geometry as needed. However, this case is relatively unlikely that it may not be worth implementing.

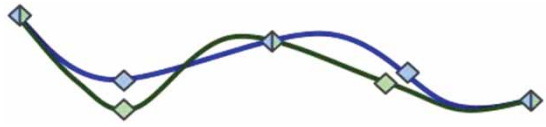

Fig. 3: Identification of unique edges by discretization.

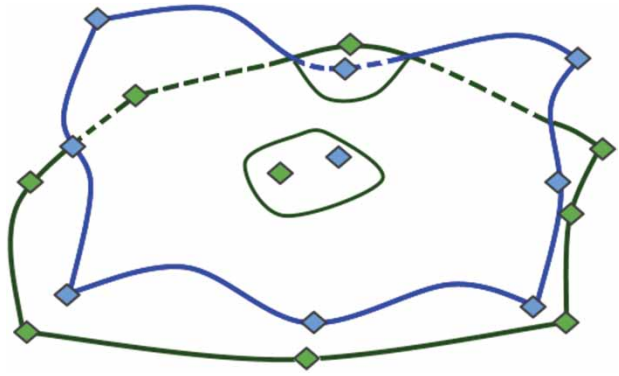

Fig. 4: Identification of unique faces by discretization.

4. IMPLEMENTATION OF METHODS

An implementation of the methods discussed above has been performed and shows promising results. These methods are presented as they are implemented into multi-user CAD systems integrated directly into the commercial CAD systems of NX and Inventor which are called NXConnect and InventorConnect respectively. Both are multi-user CAD systems which are currently being developed at the NSF Center for e-Design, BYU. We discuss the implementation of the hybrid file feature identification as it is currently implemented in InventorConnect and

NXConnect. The topology naming implementation for NXConnect will be discussed as well.

4.1. Hybrid File Feature Identification Implementation in InventorConnect

The method of using attributes to uniquely identify features has been fully implemented in InventorConnect, the multi-user prototype for Inventor. The attribute interface in Inventor had a lot to do with our decision to use attributes in the identification of different features in the part. The interface included a way to find objects based on the attributes attached to them, and also included a useful tool that allows you to query any attributes whose parents were deleted by the user. This allows for robust tracking of any deletions that occur in the part.

In the InventorConnect implementation, when a new feature is created by the user, an entry is made in the database to store this new feature. Upon creation in the database, the unique identifier from the database is written to an attribute attached to the newly created feature, establishing a permanent link between the feature and its corresponding database entry. This link persists through different edits of this feature and through multiple editing sessions in the part. Any feature that has an attached attribute linking it to a database entry is recognized by the InventorConnect plugin as a previously uploaded feature. Any newly created features not containing an attribute linking them to the database are uploaded if they are compatible with the plugin. In addition to storing a link to the database, additional attributes were stored to reflect the state of the feature and its parameters when it was stored in the database. This was done to allow for simple comparisons of the old state of the feature with the newest changes that a user has made, allowing the plugin to check for user changes without having to query the database. This essentially provides a history of the last submitted change to any given feature. This is particularly useful for implementing offline editing of multi-user CAD files. The stored history will reflect the state of the part when the user was last connected to the database. When the part and its offline changes need to be merged back into the current database version, this data will be valuable in determining which features were modified by the user and in resolving offline editing conflicts.

In addition to tracking feature creation by the user, the ability to track feature deletion through attributes was valuable. In prior architectures, deletions were easily tracked when event callbacks occurred upon deletion, but when there were no appropriate event callbacks the multi-user plugin had to check each stored pointer for each feature and determine if that feature had been deleted. With a feature that has attributes stored to it, deleting the feature does not delete the attached attributes, but flags them as being unattached from their parent features. A simple query of the unattached attributes is performed to detect deleted features, and these are then marked in the database as being deleted. This also has broader implications for offline editing of multi-user parts. When a user edits a part offline, no event callbacks are monitored by the plugin, so in order to capture user deletions, all of the features would have to be compared to the database when the user brings their part back in sync with the database. With the new method, any deletions that occur can be checked for at any time simply by examining the attributes that have become detached from their parent features.

In addition to storing database identifiers on features, InventorConnect also stores the revision number of the database as an attribute. This allows for a part that has been saved locally but that has become out of date with the database to be updated with only the changes that need to occur. Upon opening a part, all new changes to the database are loaded, rather than having to load and check every single change that has occurred in the part.

4.2. Hybrid File Feature Identification Implementation in NXConnect

Identifying features by attribute has also been fully implemented in NXConnect. In this implementation, the saved part files with the unique identifiers are actually uploaded and stored in the database, allowing users to automatically download and open the saved files when they wish to load a part. Because these files contain the unique identifier stored as attributes, they can be opened on any computer and used in NXConnect. One additional improvement in this implementation is the compatibility with the previous NXConnect architecture. NXConnect stores dictionaries which hold pointers to every object and entity in the part, including faces, edges, and bodies. This allows NXConnect to quickly search and access any needed objects in the part file.

One difficulty which was overcome during the implementation of the new loading method is that these dictionaries needed to be generated upon loading the part file to be available for future use. This meant that a reference to every object and entity needed to be accessed every time a part was opened, which was trivial for features, but time consuming for faces, edges and bodies due to the large number of these entities present in a part. In order to further reduce load times, lazy loading of these dictionaries was implemented. Upon saving a part file, the current dictionaries are serialized and stored as string attributes attached to each feature in the part. When the part is again loaded, these strings are parsed and blank dictionary entries are generated, creating a placeholder for the face, edge, and body pointers of each feature. If the user ever uses an operation requiring one of these pointers, the dictionaries are loaded on demand for the particular feature needed

by NXConnect. Upon being loaded on demand, the data is cached in the dictionary, allowing the pointers to be used again in a different context without the need to load them again.

4.3. Topology Identification Implementation

The above mentioned topology identification methods have been implemented into NXConnect using the NX Open API in three specific ways: the first uses multiple types of geometric data for the object as a whole, the second uses only positional data at multiple positions and the last is a combination of the first two. Each implementation uses the same general flow, in which geometric data from the edges, faces and bodies are saved to the database along with the feature parameters. Just after feature recreation, that data is compared against each edge, face, or body, until a match is found, at which point the object is given the name associated with that data.

The first implementation uses different types of data for different objects. For edges, that data consists of the endpoints of the edge, and the tangent vector and curvature at one of the endpoints. Faces use a face normal and the inverse of the maximum radius of curvature; to reduce the number of times that data must be calculated, only faces with the same number of edges as the target face are compared. Bodies are then identified by the positive identification of all their constituent faces and edges. The API has a bug, however, which prevents this method from being completely accurate. The method used to calculate normal vectors, tangents and curvatures (the NXOpen.GeometricAnalysis. GeometricProperties class of the .NET API) sometimes returns garbage data for objects created using the API (as opposed to through the GUI). Thus, the method returns correct data for features being saved to the database, but incorrect data when attempting to identify them after recreation.

The second implementation uses only positional data, but increases the number of samples, using the somewhat randomly chosen parameter values. For edges, this leads to four points for comparison, including the endpoints. Given the additional complexity of faces, only two points are deemed sufficient to reduce the collision probability to acceptably low levels, and as in the other method, only faces with equal numbers of edges are compared. Bodies are again identified by the constituent faces and edges. This implementation also has an API-related problem, in that the parameterization of edges and faces is sometimes different depending on whether the feature is being created or edited, or using the API vs. the GUI. Thus, two samples taken at the same parameter values from the same face on different computers may possibly have different values, and the algorithm fails to identify the face.

Since both of these implementations work much of the time, but API bugs cause them each to fail in certain cases, the last implementation is a combination of both these cases. Since both approaches are used together, it offers a safety net so that when one approach fails the other may catch it. This redundancy reduces the total number of times it fails overall and appears to be a fairly robust approach as shown in the results section of this paper.

5. RESULTS

5.1. Hybrid File Feature Identification

The implementation of the hybrid file multi-user feature identification method in InventorConnect to uniquely identify features in multi-user CAD applications has proven to vastly decrease load times. In one test, a part with 10 features took 47 seconds to load using the old method of re-creating the part feature by feature. Opening this same file in the new implementation of InventorConnect took less than 4 seconds. These results are more pronounced for larger parts and for assemblies, which can contain hundreds and thousands of features.

In the NXConnect implementation, significant results have also been observed. For one part file that had approximately 100 features, the load time was reduced from 2 minutes to 3 seconds. In another test, shown in Fig. 5, a tractor model assembly with 16 components loaded in 7 seconds, which is 1% of the 10 minute load time of the former method. Where the previous loading implementation prevented the scalability of NXConnect due to unacceptable load times, the new loading method introduces almost no additional wait time beyond the time needed to download the part. Thus it is shown that this new methodology provides a scalable approach for commercial multi-user CAD to be compatible with large, complex assemblies and parts.

In addition to improving the user experience by vastly decreasing loading times, it has allowed for a primitive implementation of offline editing, with the ability to add new features and delete features offline and have those changes uploaded the next time you connect to the database. Another major benefit is that it allows for part files created in single-user CAD to be seamlessly integrated into a multi-user CAD implementation.

5.2. Topology Identification

The persistent topology naming method has allowed complex geometry associations to be produced within NXConnect. Despite the fact that the NX API has known bugs which cause errors to occur in the NXConnect implementation for this method, the combined implementation of both approaches discussed above is shown to be reasonably robust. This is shown by the use of an internal tracking system to observe the robustness of this implementation. Over a three-week period, about 22 active users clicked at least

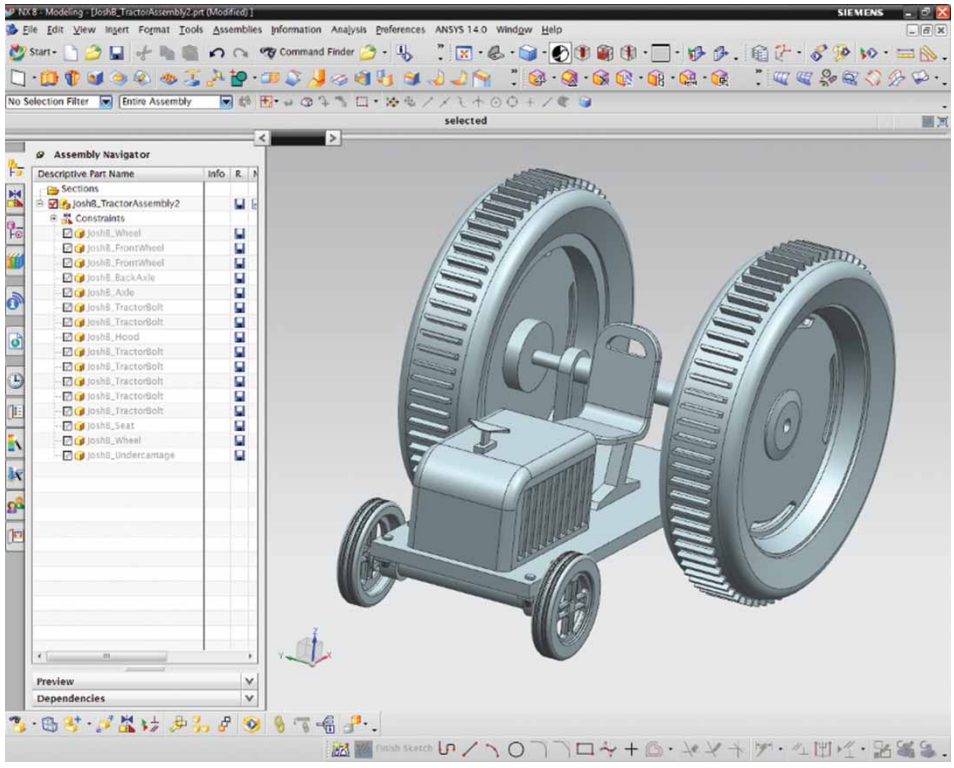

Fig. 5: Tractor assembly loaded in 7 sec. with new method vs. 10 min. with previous method.

1771 body-generating commands. Each command could generate multiple bodies, or if the user cancelled the command, no bodies. The data reflect some usage of ordinary NX, some debugging of NXConnect, and significant large-scale modeling, including the tractor demo we discuss above. During this period of use, only 22 bodies with identification problems were created. Given reasonable assumptions (most of the usage was in NXConnect, and an average of one body was generated per body-generating modeling command), this suggests the topology naming scheme is on the order of 99% effective. These numbers reflect the effectiveness of the naming methodology despite known API bugs associated with these implementations; resolving these should further reduce issues related to topology identification.

6. CONCLUSIONS

The newly developed method to associate CAD objects directly to the database using CAD object attributes has shown to increase part and assembly scalability in multi-user CAD tools by dramatically reducing part and assembly load time. This time reduction grows significantly as object count increases in parts and assemblies. It has also allowed for offline editing which enables users to add and delete features offline and have those changes uploaded the next time they connect to the database. Another major benefit is that the methodology integrates single-user commercial CAD parts into a multi-user CAD implementation.

The persistent topology naming methodology theoretically solves the problem of multiple users referencing the same topology entity on separate clients. Due to limitations in the NX API, the implementation has required additional redundancy to be implemented effectively, but has shown to be robust enough to support modeling of complex parts and assemblies. The results of these implementations have been very promising for the future development of multi-user solutions for commercial CAD.

ACKNOWLEDGEMENTS

Special thanks to all the industry sponsors participating in the Center for eDesign, BYU who funded this research: Boeing, Pratt & Whitney, Belcan, PCC Airfoils, Spirit Aero Systems and CD-adapco.

REFERENCES

[1] Bidarra, R.; Nyirenda, P.; Bronsvoort, W.: A feature-based solution to the persistent naming problem, Computer-Aided Design and Applications, 2(1), 2005, 517–526.

[2] Bidarra, R.; Bronsvoort, W.: Persistent naming through persistent entities; Geometric Modeling and Processing, 2002. Proceedings, 2002; 233–240.

[3] Bidarra, R.; E. van den Berg; W. F. Bronsvoort.: A Collaborative Feature Modeling System, Journal of Computing and Information Science in Engineering, 2(3), 2002, 192, http://dx.doi.org/10.1115/1.1521435

[4] Cannon, L.; Nysetvold, T.; Phelps, G.; Winn, J.: Jensen C. G.: How Can NX Advanced Simulation Support Multi-User Design?, Computer Aided Design and Applications, PACE Vol. 2, 2012, 21–32.

[5] Capoyleas, V.; Chen, X.; Hoffmann, C.: Generic naming in generative, constraint-based design; Computer-Aided Design, 28(1), 1996, 17–26, http://dx.doi.org/10.1016/0010-4485(95)00014-3

[6] Chen, X.; Hoffmann, C.: On editability of feature-based design; Computer-aided design, 27(12), 1995, 905–914, http://dx.doi.org/10.1016/0010-4485(95)00013-5

[7] Chen, Z.; Gao, S.; Zhang, F.; Peng, Q.: An approach to naming and identifying topological entities; Chinese Journal of Computers, 24(11), 2001, 1170–1177.

[8] Jing, S.; F. He; S. Han; X. Cai; and H. J. Liu.: A method for topological entity correspondence in a replicated collaborative CAD system, Computers in Industry, 60(7), 2009, 467–475, http://dx.doi.org/10.1016/j.compind.2009.02.005

[9] Jing, S.; He, F; Cai, X; Liu, H.: Collaborative naming for replicated collaborative solid modeling system; International Design Engineering Technical Conferences & Computers and Information in Engineering Conference, 2008, 141–150.

[10] Kripac, J.: A mechanism for persistently naming topological entities in history-based parametric solid models, ACM Symposium on Solid Modeling and Applications, 3, 1995, 21–30, http://dx.doi.org/10.1145/218013.218024

[11] Marshall, F.: Model Decomposition and Constraints to Parametrically Partition Design Space in a Collaborative CAx Environment, Brigham Young University, Master's Thesis, 2011.

[12] Moncur, R.; Jensen, C.; Teng, C.; Red, E.: Data Consistency and Conflict Avoidance in a Multi-User CAx Environment, Computer-Aided Design and Applications, 10(5), 2013, 727–744.

[13] Qiang, L.; Y. F. Zhang; and a. Y. C. Nee.: A Distributive and Collaborative Concurrent Product Design System through the WWW/Internet, The International Journal of Advanced Manufacturing Technology, 17(5), 2001, 315–322, http://dx.doi.org/10.1007/s001700170165

[14] Ramani, K.; A. Agrawal; M. Babu; C. Hoffmann.: CADDAC: Multi-Client Collaborative Shape Design System with Server-based Geometry Kernel, Journal of Computing and Information Science in Engineering, 3(2), 2003, 170, http://dx.doi.org/10.1115/1.1582882

[15] Red, E.; Jensen, G.; Holyoak, V.; Marshall, F.; Xu, Y.: v-Cax: A Research Agenda for Collaborative Computer-Aided Applications, Computer-Aided Design and Applications, 7(3), 2010, 387–404.

[16] Red, E.; Jensen, C.; French, D.; Weerakoon, P.: Multi-User Architectures for Computer-Aided Engineering Collaboration. International Conference on Concurrent Enterprising, 2011.

[17] Shaojin, S.; Jianjun, C.; and Jindou, L.: An Asynchronous CAD Collaborative Design Model, 2010 International Conference on Computer Application and System Modeling, 6, 2010, 563–568, http://dx.doi.org/10.1109/ICCASM.2010.5620677

[18] Siemens Corp., Parasolid Documentation.

[19] Wu, J.; Zhang, T.; Zhang, X.; Zhou, J.: A face based mechanism for naming, recording and retrieving topological entities; Computer-Aided Design, 33(10), 2001, 687–698, http://dx.doi.org/10.1016/S0010-4485(00)00099-3

[20] Xu, Y; E. Red, E.; Jensen, G.: A Flexible Context Architecture for a Multi-User GUI, Computer-Aided Design and Applications, 8(4), 2011, 479–497, http://dx.doi.org/10.3722/cadaps.2011.479-497

[21] Zhou, X.; Li, J.: A Web-based synchronized collaborative solid modeling system, Chinese Journal of Computer Integrated Manufacturing Systems, 2003, 960–965.