COMPUTER-AIDED DESIGN & APPLICATIONS, 2017
VOL. 14, NO. 3, 284–292
http://dx.doi.org/10.1080/16864360.2016.1240449

Computer-AidedDesign Taylor & Francis
Taylor & Francis Group

Pseudo-singleton pattern and agnostic business layer for multi-engineer, synchronous, heterogeneous CAD

K. Eric Bowman, C. Greg Jensen and Devin Shumway

Brigham Young University, USA

ABSTRACT
Product development processes are currently highly serial workflows that prevent needed cycle time reduction. Multi-user synchronous CAD is already improving the parallelization of these workflows and the next step is to facilitate multi-user synchronous heterogeneous CAD. This paper proposes a pseudo-singleton design pattern and a class signature which are necessary in the business logic layer of an application architecture that affects multi-user synchronous collaboration across heterogeneous CAD clients.

KEYWORDS
Heterogeneous CAD; Neutral Format; MESH CAD

1. Introduction

Engineering companies are sociotechnical systems in which engineers, designers, analysts, etc. use a wide array of software tools as they follow prescribed product development processes. The purpose of these amalgamated systems is to develop new products as quickly as possible while maintaining quality as well as meeting customer and market demands. Task speed up in a parallelized system can be modeled by Amdahl's law and so is governed by how much of a process can be parallelized [1]. Researchers at Brigham Young University have shortened engineering design cycle times through the development of synchronous collaborative CAD tools [10], [12], [17], [20]. Other research teams have shortened design cycle times by extending seamless interoperability across heterogeneous design tools and domains [3]–[6], [8], [9], [13]–[15], [18], [19], [22]–[24]. Multi-engineer synchronous (MES) collaboration across heterogeneous CAD environments is the focus of this paper. A logical architecture that supports both MES collaboration and interoperability is defined and tested for robustness and proposed as the start of a new standard for interoperability. In particular, a pseudo-singleton pattern is proposed to ensure data stability despite unordered data and a multi-engineer synchronous heterogeneous (MESH) object class pattern is proposed to allow heterogeneous clients to interoperate even if the server has no knowledge of the client. This architecture has demonstrated design and modeling interoperability between Siemens' NX, PTC's Creo and Dassault Systemes' CATIA CAD applications and interoperability between Siemens' NX and Dassault Systemes' CATIA are specifically demonstrated in this paper. The 2D point, 2D line, 2D arc, 2D circle, 2D spline, 3D point, extrude, and revolve features have been demonstrated. Complex models have successfully been modeled and exchanged in real time across heterogeneous CAD clients and have validated the architectural approach proposed for MESH CAD data storage.

2. Related work

This section first discusses the background of multi-engineer synchronous (MES) design applications. Then it discusses multi-engineer synchronous heterogeneous (MESH) applications including the Neutral Parametric Database (NPDB) standard, the data layer used in this MESH application. Finally it will review the singleton pattern, from the field of computer science.

2.1. *Multi-Engineer synchronous CAD*

Jensen, Red et al. [20] developed NXConnect which provides real-time MES modeling between NX clients. This technology is currently under commercialization [10]–[12]. Cai created a multi-user SolidWorks experience using similar ideas to NXConnect [2]. Maher did similar research using AutoCAD [16]. While these systems support concurrent collaboration within a homogeneous CAD environment, they do not support a synchronous heterogeneous CAD design environment.

CONTACT K. Eric Bowman ericbowman@gmail.com; C. Greg Jensen cjensen@byu.edu; Devin Shumway devin.shumway@gmail.com

The ideal CAD environment would support concurrent collaboration with the union of CAD features across heterogeneous CAD systems. The purpose of this research is to define a neutral format which merges the principles used to create a MES CAD system with those used to create a heterogeneous CAD system to create a new MESH architecture.

2.2. Neutral parametric database

Bowman et al. have developed a neutral parametric database (NPDB) that serves as a neutral storage format for CAD data. [7] It fulfills key requirements that previous neutral formats have not. First, it is based off of the mathematical definition of parametric features rather than the CAD system's proprietary definition. Second, the neutral parametric database is normalized to prevent update anomalies. Third it has been shown to be compatible with an object oriented class structure mapping and finally, it was implemented using industry best practice tools and methods. Later Bowman et al. improved the NPDB through multi-reference interface inheritance. [21] This architectural improvement allows for the storage of ambiguous CAD feature references while still maintaining referential integrity. This paper discusses key issues that were solved in order to create a full MESH CAD application with the NPDB as its foundation.

2.3. The singleton pattern

From mathematics set theory we learn a Singleton set has only one element. For example, the set containing whole numbers less than one (1), or the set of all integers that are neither positive nor negative. The axiom of regularity was introduced by von Neumann in 1925 and is one of the axioms of Zermelo-Fraenkel set theory and guarantees that no set is an element in itself. In the mid 90′s computer scientists began defining and publishing what they called design patterns, with one being the singleton pattern. In the book Design Patterns [25], it defines the singleton pattern as “a class only [having] one instance, and [providing] a global point of access.” Or in other words, the singleton pattern in software engineering is a reusable design solution where the instantiation of a class is limited to the creation of one object. This is done primarily to eliminate the use of global objects and variables but still allow the programmer to access static data. It also allows the user to implement interfaces which allows them to pass the singleton as an object into functions. This ability is the main difference that separates the singleton pattern from a static class. Much of the singleton pattern code uniqueness stems from it close association to category theory and the creation of formal structures. An example of the instantiation of a singleton class can be seen in Listing 1.

```
public class Singleton
{
    private static Singleton Instance { get; set; }

    private Singleton() { }

    public static Singleton GetInstance()
    {
        if (Instance == null) Instance = new Singleton();

        return Instance;
    }
}
```

Listing 1. The Singleton Pattern

3. Method

Both MES homogeneous CAD and single-user CAD translation software exist in the literature, however there has not been MESH CAD software demonstrated. In order to extend multi-user homogeneous CAD software there are a number of problems that must be solved. First was the development of a new data storage standard capable of storing heterogeneous CAD data easily while avoiding update anomalies. This has been developed by Bowman et al. and is known as the NPDB. Another two key problems are first that a MESH client must be able to map a flat list of feature commands to an associative feature tree-structure and second that the system be client-agnostic.

3.1. The singleton pattern

In order for a system to support parametric CAD models properly, it must represent them as a directed acyclic graph (DAG) where the nodes are features and the edges are parent-child relationships between features as shown in Fig. 1.

This is an important task in CAD client-server architectures because computer queries typically return a list of features with no regard for their dependencies. The listed features not only don't clearly display dependencies but they can also be returned in a random order. For example, if you queried all of the features from Fig. 1 the result might look like the following numbered list:

1. Feature 2
2. Feature 3
3. Feature 4
4. Feature 1

Unfortunately, the part model cannot be built in this order because feature 4 depends on features 2 and 3,

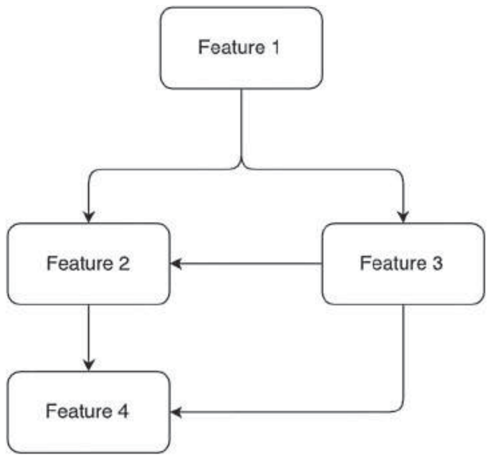

Fig 1. CAD Data should be represented as a Directed, Acyclic Graph

feature 3 depends on feature 1, and feature 2 depends on feature 1. So, the order this model must be built is as follows:

1. Feature 1
2. Feature 3
3. Feature 2
4. Feature 4

Any other build order would result in a fatal application exception. Finding the correct build order for a part is a critically task that is subject to algorithm pitfalls.

The intuitive way to solve a problem like this is to explicitly find and maintain the correct feature order using consistency managers. Each part could have an overall consistency manager which manages dependencies of all features within the part. This manager could be made more efficient by dividing features into logical groups in a divide-and-conquer style algorithm. All part-level consistency managers in a session would need to be managed by an overall consistency manager. This type of architecture is shown in Fig. 2.

This method would function for a time but has several drawbacks. Specifically, the logic for this type of architecture is difficult to develop and must be changed whenever feature definitions are modified. While the well-known topological sort algorithm has an algorithmic complexity of $O(|V| + |E|)$, typical consistency managers such as this reach nowhere near that ideal speed because each edge traversal is an algorithm in and of itself that can even include a database query. Additionally, previously developed MES databases did not explicitly store references and so the dependency graph must be re-assembled for each feature, so loading a part in these systems has an algorithmic complexity of $O(|V|^2 + |E||V|)$). Thus, maintaining CAD data graphs using a consistency manager adds significant overhead. This approach is unlikely to scale to a large assembly with hundreds of thousands of parts and millions of features.

Another way to solve this issue is through the pseudo-singleton pattern which is a development from the singleton pattern discussed in the section 3.1. Because all reference are explicitly stored in the neutral parametric database, the resolution of each dependency the first time any referenced feature is used is $O(1)$, the speed to query a single record from a database. The resolution of any later reference that uses that feature is once again O(1) because it is a simple dictionary lookup. Although the algorithmic complexity of a database query is the same as that of a dictionary lookup, dictionary lookups are

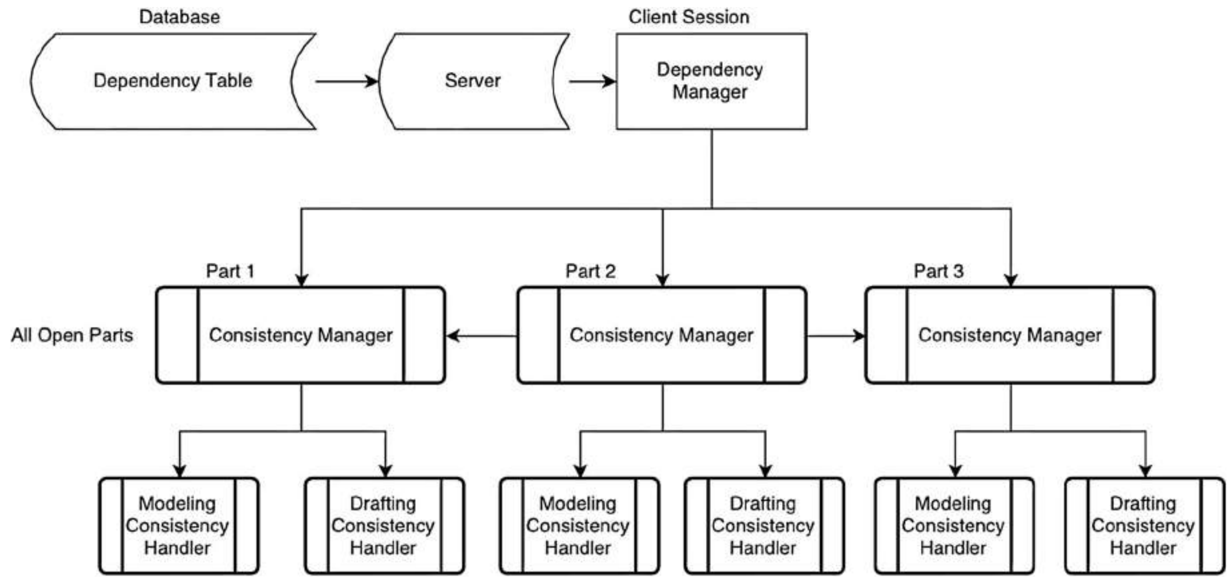
Database
Dependency Table
Server
Client Session
Dependency Manager
Part 1
Part 2
Part 3
All Open Parts
Consistency Manager
Consistency Manager
Consistency Manager
Modeling Consistency Handler
Drafting Consistency Handler
Modeling Consistency Handler
Drafting Consistency Handler
Modeling Consistency Handler
Drafting Consistency Handler

Fig 2. Consistency Manager Architecture to Ensure CAD Part Consistency.

faster in practice and thus preferred. In addition to its high speed, the pseudo-singleton pattern is preferred to a consistency manager due to its ease of use and maintenance. All feature classes can implement the same pattern which automatically performs all dependency resolution without any further code development.

The pseudo-singleton pattern makes its class constructor private and only allows uniquely identified instances of itself to be queried. If the uniquely identified instance exists it is returned from a static dictionary. If the instance has not been created the class queries the server for the instance, adds it to the static dictionary and returns it. In this way feature instances can be used by a client developer without taking concern for their associations. In other words, any feature built from the pseudo-singleton pattern maintains its own dependencies with no further logic. Sample code that illustrates this pattern is shown in Listing 2.

```
public class PseudoSingleton
{
    private static Dictionary<Guid, PseudoSingleton> Instances;

    private PseudoSingleton(Guid Id) { }

    public static PseudoSingleton GetInstance(Guid Id)
    {
        if (!Instances.ContainsKey(Id))
            Instances.Add(Id, new PseudoSingleton(Id));

        return Instance[Id];
    }
}
```

Listing 2. The Pseudo-Singleton Pattern

The utility of this pattern can be demonstrated by assigning features to the DAG shown in Fig. 1. Below in Fig. 3 is a DAG representation of a part containing a coordinate system (CSYS), a 3D Point built from that CSYS,

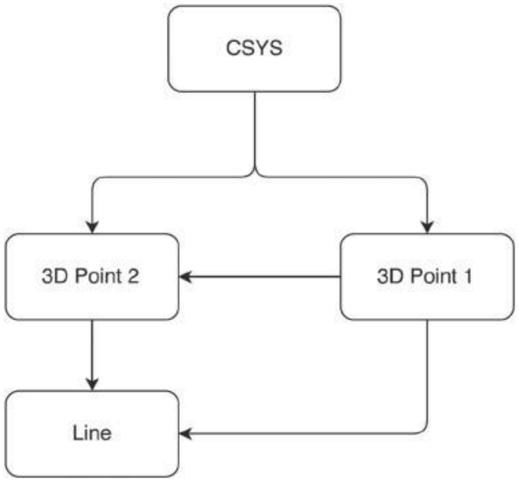

Fig 3. Sample CAD Part DAG.

another point built from the CSYS and relative to the first point, and a line connecting the two points.

Assigning those features to the queried feature list mentioned above we would get the following list of feature messages:

1. 3D Point 2
2. 3D Point 1
3. Line
4. CSYS

The first feature, *3D Point 2* has a dependency on *CSYS* and a dependency on *3D Point 1*. When this message is processed, *3D Point 2* calls the "GetInstance" method, passing in *CSYS's* GUID. Since *CSYS* is not yet in the feature dictionary its constructor is called, it is added to the feature dictionary and it is returned to *3D Point 2*. *3D Point 2* then calls the "GetInstance" method passing in *3D Point 1′s* GUID Since *3D Point 1* is not yet in the feature dictionary its constructor is called. *3D Point 1* is built off of *CSYS* so *3D Point 1′s* constructor calls the "GetInstance" method passing in *CSYS's* GUID. Since *CSYS* is already in the feature dictionary it is simply returned. *3D Point 1* is then added to the feature dictionary and returned. The dependency graph for *3D Point 2* has automatically been assembled. The point is created and added to the feature dictionary.

The second feature, *3D Point 1* has already been placed in the feature dictionary so when the loading algorithm reaches its message it simply verifies that it is already in the feature dictionary and moves on.

The third feature, *Line* has a dependency on *3D Point 1* and *3D Point 2*. When this message is processed, *Line* calls the "GetInstance" method, passing in *3D Point 1′s* GUID. Since *3D Point 1* is already in the feature dictionary it is simply returned. The same happens for *3D Point 2* since it is also in the feature dictionary. The dependency graph for *Line* has automatically been assembled. The line is created and added to the feature dictionary.

The final feature, *CSYS,* has already been placed in the feature dictionary so when the loading algorithm reaches its message it simply verifies that it is already in the feature dictionary and moves on.

Next an implementation of this pattern will be shown. First all CAD feature classes should inherit from a base class which contains a dictionary of all features. As shown in Listing 3.

Since all features will inherit from this base class they have access to this dictionary. Note also the utility of using inheritance to ensure that every feature will have a GUID to uniquely identify it which matches the GUID primary key in the NPDB. There is also a DeleteFromClient method that is fully implemented in this base class

```
public abstract class MUObject
{
    public Guid GUID…
    private Guid guid;

    public static Dictionary<Guid, MUObject> MUFeatures { get; set; }

    public virtual void DeleteFromClient() …

    public abstract void DeleteFromServer(MUObject ObjectToDelete, CADTypes CADType);
}
```

Listing 3. Feature Dictionary Implementation for Pseudo-Singleton Pattern.

```
public class MUPoint : MUObject

    //Properties
    public override DBInteropObject serverObject

        get { return ServerPoint; }

    public DBFeature ServerPoint { get; private set; }
    public HybridShapePointCoord CATIAPoint { get; private set; }

            Public
    /// <summary>
    /// Creates CAD specific point from the server point and CAD system
    /// </summary>
    public static MUPoint GetInstanceFromServer(DBFeature Point, CADTypes CADType)…
    /// <summary>
    /// This method checks to see if the CATIA point is already in the MUFeatures
    /// dictionary.  If it is it's returned, if not it's created
    /// </summary>
    public static MUPoint GetInstanceFromCATIA(HybridShapePointCoord clientPoint,
                                                Guid partGUID)…
    /// <summary>
    /// Use this method when the ServerVersion has changed.
    /// </summary>
    public void UpdateFromServer(DBFeature serverPoint, CADTypes CADType)…
    /// <summary>
    /// Use this method when the CATIAVersion has changed.
    /// </summary>
    public void UpdateFromCATIA(HybridShapePointCoord ClientPoint)…

    public override void DeleteFromServer(MUObject ObjectToDelete, CADTypes CADType)…


    //Private…
```

Listing 4. 3D Point Implementation.

and a DeleteFromServer method signature that all subtypes must implement. Pertinent portions of the 3D Point class from the CATIA client are shown in Listing 4.

The reason that all of these methods are in the class will be discussed in the next section, but the pseudo-singleton pattern is implemented in the GetInstanceFromCATIA and the GetInstanceFromServer methods. The GetInstanceFromServer method is shown in Listing 5.

Any other place the application needs to reference a point, it must do so through this method rather than creating the point. This can be seen in the Sphere implementation shown in Listing 6.

Note that when the sphere center is used, rather than perform any logic checks or take concern whether the center point has been received yet or not, the code simply grabs the point of interest. If the point has been created already it will simply be returned from the dictionary. If the point has not been created it will be automatically created in the correct dependency order. Because of this pattern it does not matter if the point message or the sphere message is sent from the server first, the sphere

```
/// <summary>
/// Creates CAD specific point from the server point and CAD system
/// </summary>
public static MUPoint GetInstanceFromServer(DBPoint Point)
{
    //If the server point already exists in the dictionary, it is returned
    if (MUFeatures.ContainsKey(Point.GUID))
        return (MUPoint)MUFeatures[Point.GUID];

    //If the point doesn't exist it is created
    MUPoint result = new MUPoint();
    result.CreateCATIAPoint(Point);

    //The newly created MUPoint object is added to the MUFeatures dictionary
    MUFeatures.Add(result.GUID, result);

    return result;
}
```

Listing 5. Pseudo-Singleton Implementation in 3D Point.

```
/// <summary>
/// Creates a CATIA Sphere
/// </summary>
/// <param name="serverPoint"></param>
void CreateCATIASphere(DBFeature serverSphere)
{
    HybridShapeSphere sphere;
    GUID = serverSphere.GUID;
    ServerSphere = serverSphere;

    HybridShapePointCoord center =
                (MUPoint.GetInstanceFromServer(serverSphere.CenterPoint)).CATIAPoint;

    PartDocument partDoc = CurrentPart.CATIAPart;

    HybridShapeFactory hybridShapeFactory =
                (HybridShapeFactory)partDoc.Part.ShapeFactory;
    partDoc.Part.InWorkObject = MUObject.currentBody;

    INFITF.Reference centerRef = partDoc.Part.CreateReferenceFromObject(center);

    sphere = hybridShapeFactory.AddNewSphere…
    sphere.Limitation = 1;
    sphere.set_Name(serverSphere.Name);
        Utilities.SetGuid(sphere, GUID);

    MUObject.currentBody.InsertHybridShape(sphere);

    partDoc.Part.Update();

    CATIASphere = sphere;
    ServerSphere = serverSphere;
}
```

Listing 6. Sphere Implementation Taking Advantage of the Pseudo-Singleton Pattern.

will assemble its own dependency graph in the correct order.

3.2. *Agnostic business layer*

In addition to working with unordered feature data, a heterogeneous server should work the same irrespective of the client that is connected to it. From a data storage perspective this has been addressed by creating a neutral parametric database (NPDB) which maintains referential integrity, thus preventing data corruption. From the server perspective this is further enforced by requiring all clients to use a standard set of messages. From a client perspective there must be a standard, documented architecture that any developer can use to integrate their client with the heterogeneous server.

In this case the NPDB was mapped to server-side classes which enforce full referential integrity of the database. Once those classes were made, a standard set of methods were created which perform the standard create, read, update and delete (CRUD) operations both from the client to the server and from the server to the client. The standard methods use the mapped classes as arguments and the server. Inheritance can be used to enforce that any client implement the exact set of methods with the correct arguments. Since data integrity is enforced by those server objects, any client that uses the standard set of methods can fully interact with the heterogeneous server architecture. The standard set of methods can be seen in Listing 3 above and the delete methods are enforced in the proposed way as shown previously in Listing 4.

4. Results

A logic architecture that enables a CAD client-server architecture to work with unordered feature data and irrespective of which commercial CAD client or clients are being used was successful defined. This architecture was realized in a prototype implementation that consists of a database, communication, logic, and client tiers in a standard N-tiered configuration. Modeling data is stored in a NPDB using a TPT mapping method. When modeling changes are made to the database, the communication tier handles messages to and from the server, client, and database. The logic tier contains classes with methods for creating and updating features bi-directionally between client and server. During a MESH session the client tier tracks users' actions and passes features to the logic tier for neutralization and storage.

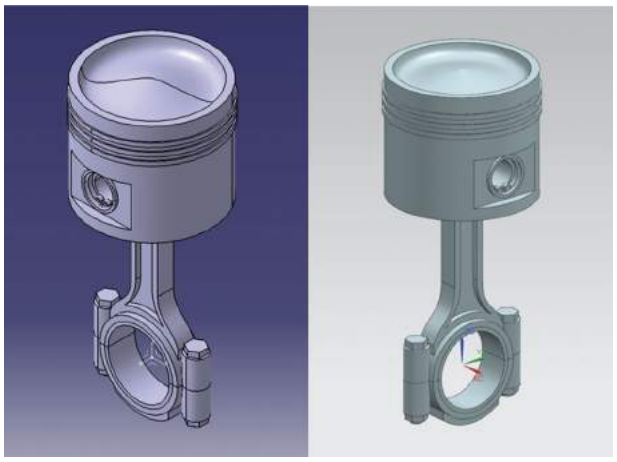
Fig 4. Piston and connecting rod assembly modeled in a MESH CAD environment.

To date, this architecture supports concurrent collaboration between the NX, CATIA, and Creo CAD systems. While our initial efforts focused on interoperability between NX and CATIA, the two dominant commercial CAD packages used by large multi-national companies, the Creo MESH client is now in the prototype testing phase. The 2D point, 2D line, 2D arc, 2D circle, 2D spline, 3D point, extrude, and revolve features were developed and implemented. Many different modeling tests and sessions have been successfully completed, i.e. heterogeneous clients have successfully participated in the creation and editing of a NPDB models. One such example is the piston and connecting rod assembly, seen in Fig. 4, which was modeled by four users on four different computers. Two users modeled using NX clients and the other two used CATIA clients. All the features implemented in this research were used in the modeling of the piston and connecting rod.

During all tests, including the piston and connecting rod, users worked simultaneously creating features on their own CAD client. For example, as one user exits a sketch, changes made by other users are pulled to this client's session in real time, and the created sketch is pushed to all other clients so that the model on all screens stays in sync. Semantic and syntactic conflicts have been studied, architected, implemented and tested, and are preventable using Hepworth et al. methods [11]. Another example of MESH implementation is seen in Fig. 5. This shows a collaborative creation/editing session between one NX user, one CATIA user and one Creo user. All are working simultaneously in the same NPDB session. They can each see, in real-time, the modeling and editing

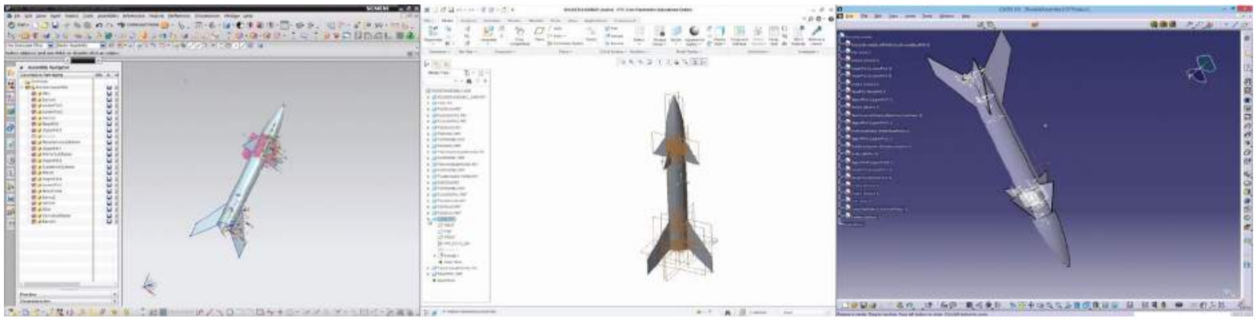
Fig 5. A CATIA user, NX user and Creo user collaborating on a rocket assembly.

operations being completed by the other collaborators, reducing design turn-backs and eliminating all delays normally incurred from IGES and/or STEP translations.

On a daily basis students are modeling and designing using MESH. These concerted efforts are intended to test and debug the use of the pseudo-singleton patterns with in the frame work of the MESH and its NPDB implementation. Additional geometric features and interfaces are being added to both the MESH and NPDB.

5. Conclusions

Industrial partners of the NSF Center for e-Design have funded and supported this research seeking tools and methods to assist in the development of high-quality products as quickly as possible. Many of these partners employ multiple CAD system, if not internally, they must interface with them across their supply chain. The pseudo-singleton pattern supported within the MESH CAD system which interfaces with the NPDB has enabled, and when fully implemented, will shorten their product development lifecycle. Engineering design methods will become more efficient, enabling companies to reduce non-recurring costs without design quality losses due to a lack of CAD interoperability. As a compliment to homogeneous multi-user CAD this research has clearly demonstrated the viability of heterogeneous multi-user CAD across Siemens NX, Dassault Systemes CATIA and PTC Creo. Yes, other researchers of CAD interoperability have worked toward similar goals. However, this effort demonstrates the possibilities of true seamless multi-user synchronous modeling between heterogeneous engineering CAD clients.

A functional prototype MESH CAD client-server architecture with logic rules to support unordered feature data and a client-agnostic server was developed, tested and shown to be stable. Moderately complex modeling tasks were completed by small teams using the prototype in order to demonstrate that the client server approach taken and the NPDB data storage method are functionally sound. While a core set of features were chosen for implementation that allow for non-trivial model generation, the authors clearly disclose that an expanded "full-feature set" will be required to accommodate the demands industry will place on MESH CAD. The concept of a MESH CAD system has been proven.

Acknowledgements

Research of this type does not happen in a vacuum. We would like to thank those who have worked on this problem before us because their ideas were inspirational. We would like to thank all of the members of the BYU CADLab and Dr. Tew for his contributions to our server. We could not have done this without Raytheon Missile Systems or our members in the National Science Foundation center for e-Design: Belcan, Boeing, PCC Airfoils, Pratt & Whitney, CD-adapco, UTC Aerospace Systems, and ARDEC.

ORCiD

K. Eric Bowman http://orcid.org/0000-0003-2940-8897
Greg Jensen http://orcid.org/0000-0003-1824-0945
Devin Shumway http://orcid.org/0000-0002-5186-7857

References

[1] Amdahl, G. M.: Validity of the single processor approach to achieving large scale computing capabilities, AFIPS spring joint computer conference, 1967, 187–196.
[2] Cai, X.; Li, X.; He, F.; Han, S.; Chen, X.: Flexible Concurrency Control for Legacy CAD to Construct Collaborative CAD Environment, Journal of Advanced Mechanical Design, Systems, and Manufacturing, 6(3), 2012, 324–339. http://dx.doi.org/10.1299/jamdsm.6.324
[3] Chen, X. C. X.; Li, M. L. M.; Gao, S. G. S.: A Web services based platform for exchange of procedural CAD models, Proceedings of the Ninth International Conference on Computer Supported Cooperative Work in Design, 2005, 605–610.
[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), 2002, 13–21.
[5] Dou, W.; Song, X.: Operation Command Transformation of Synchronized Collaborative Design Upon Heterogeneous CAD Systems, Journal of Algorithms & Computational Technology, 7(4), 2013, 423–448. http://dx.doi.org/10.1260/1748-3018.7.4.423
[6] Dou, W.; Song, X.; Zhang, X.: A language of neutral modeling command for synchronized collaborative design among heterogeneous CAD systems, 2009 1st International Conference on Information Science and Engineering, 2009, 12–15.
[7] Freeman, R. S.; Bowman, K. E.; Staves, D. R.; Red, D. E.: Neutral Parametric Canonical Form for 3D and 3D Wireframe CAD Geometry, ASME 2015 International Mechanical Engineering Congress & Exposition, 2015.
[8] Ganapathi, S.: A Software Model for Interoperability, The University of Texas at Arlington, 2002.
[9] Gu, H.; Chase, T. R.; Cheney, D. C.; Bailey, T.; Johnson, D.: Identifying, Correcting, and Avoiding Errors in Computer-Aided Design Models Which Affect Interoperability, Journal of Computing and Information Science in Engineering, 1(2), 2001, 156–166. http://dx.doi.org/10.1115/1.1384887
[10] Hepworth, A. I.; Nysetvold, T.; Bennett, J.; Phelps, G.; Jensen, C. G.: Scalable Integration of Commercial File Types in Multi-User CAD, Computer-Aided Design and Applications, 11(4), 2014, 459–467. http://dx.doi.org/10.1080/16864360.2014.881190
[11] Hepworth, A. I.; Tew, K.; Nysetvold, T.; Bennett, M.; Jensen, C. G.: Automated Conflict Avoidance in Multiuser CAD, Computer-Aided Design and Applications, 11(2), 2014, 141–152. http://dx.doi.org/10.1080/16864360.2014.846070

[12] Hepworth, A.; Tew, K.; Trent, M.; Ricks, D.; Jensen, C. G.; Red, W. E.: Model Consistency and Conflict Resolution With Data Preservation in Multi-User Computer Aided Design, Journal of Computing and Information Science in Engineering, 14(2), 2014. http://dx.doi.org/10.1115/1.4026553

[13] Iyer, G. R.: Development of API-Based Interfaces to Enable Interoperability Between CAD Systems During Design Collaboration, The University of Texas at Arlington, 2001.

[14] Leach, L. M.: A Language Interface for Data Exchange Between Heterogeneous CAD/CAM Databases, Rensselaer Polytechnic Institute, 1983.

[15] Li, M.; Yang, Y.; Li, J.; Gao, S.: A preliminary study on synchronized collaborative design based on heterogeneous CAD systems, Cooperative Work in Design, 2004, 255–260.

[16] Maher, M. L.; Rutherford, J. H.: A model for synchronous collaborative design using CAD and database management, Research in Engineering Design, 1997, 85–98. http://dx.doi.org/10.1007/BF01596484

[17] Moncur, R. A.; Jensen, C. G.; Teng, C. C.; Red, E.: Data consistency and conflict avoidance in a multi-user CAx environment, Computer-Aided Design and Applications, 10(5), 2013, 727–744. http://dx.doi.org/10.3722/cadaps.2013.727-744

[18] Mun, D.; Han, S.; Kim, J.; Oh, Y.: A set of standard modeling commands for the history-based parametric approach, Computer Aided Design, 35(13), 2003, 1171–1179. http://dx.doi.org/10.1016/S0010-4485(03)00022-8

[19] Rappoport, A.: An architecture for universal CAD data exchange, Proceedings of the eighth ACM symposium on Solid modeling and applications, 2003, 266–269. http://dx.doi.org/10.1145/781606.781648

[20] Red, E.; Jensen, G.; French, D.; Weerakoon, P.: Multi-user architectures for computer-aided engineering collaboration, 2011 17th International Conference on Concurrent Enterprising, 2011, 1–10.

[21] Staves, D. R.; Bowman, K. E.; Freeman, R. S.; Red, D. E.: Multi-Reference Interface Inheritance for Concurrent CAD Interoperability Applications, ASME 2015 International Mechanical Engineering Congress & Exposition, 2015.

[22] Sun, L. J.; Ding, B.: Heterogeneous CAD data exchange based on cellular ontology model, WRI World Congress on Software Engineering, 2009, 46–50. http://dx.doi.org/10.1109/WCSE.2009.106

[23] Tessier, S.: Ontology-Based Approach To Enable Feature Interoperability Between Cad Systems, Georgia Institute of Technology, 2011.

[24] Zhang, X.; W. Dou: An approach of constructing neutral modeling command set of synchronized collaborative design upon heterogeneous CAD systems, International Conference on Management and Service Science, 2009, 0–3. http://dx.doi.org/10.1109/icmss.2009.5302072