ACM Symposium on Solid Modeling and Applications (2004)
G. Elber, N. Patrikalakis, P. Brunet (Editors)

Integrated Feature-Based and Geometric CAD Data Exchange

Steven Spitz† and Ari Rappoport‡

Abstract

Data exchange between CAD systems is an extremely important solid modeling concept, fundamental both for the theory of the field and for its practical applications. The two main data exchange (DE) paradigms are geometric and parametric DE. Geometric DE is the ordinary method, in which the boundary representation of the object is exchanged. Parametric (or feature-based) DE is a novel method where, given a parametric history (feature) graph in a source system, the goal is to construct a graph in the target system that results in similar geometry while preserving as much parametric information as possible. Each method has its uses and associated problems.

In this paper, we introduce Geometry Per Feature (GPF), a method for integration of parametric and geometric data exchange at the single part (object) level. Features can be exchanged either parametrically or geometrically, according to user guidelines and system constraints. At the target system, the resulting model is represented using a history tree, regardless of the amount of original parametric features that have been rewritten as geometric ones. Using this method we maximize the exchange of overall parametric data and overcome one of the main stumbling blocks for feature-based data exchange.

Categories and Subject Descriptors (according to ACM CCS): D.2.12 [Interoperability]: data mapping; I.3.5 [Computational Geometry and Object Modeling]: Breps, CSG, solid, and object representations, geometric languages and systems; I.3.6 [Methodology and Techniques]: graphics data structures and data types, languages, standards;

1. Introduction

Data exchange (DE) is a central problem in geometric and solid modeling. It is important in practice because it is the major way for achieving interoperability [Rappoport03]. It is important for the theory of the field because it sheds light on how representations can be well defined and because it is a representation conversion problem.

The established approach for data exchange, both in theory and in practice, is geometric DE, where the boundary representation (Brep) of the object is transferred from a source to a target system. The dominant way to do this today is through the STEP, IGES and VDA standards [Bloor95]. Geometric DE is rather reliable, although in many cases it results in non-solids (unstitched boundaries) due to geometric tolerancing problems. However, its main drawback is the fact that it does not support today's most common design paradigm, feature based design, and hence is lacking in terms of its support for collaborative engineering.

All modern CAD systems are based on the feature based (FB) design paradigm, also called parametric design and history based design [Hoffmann93]. Feature based data exchange (FBDE) is thus highly desirable. In FBDE, given a parametric history (feature) graph in a source system, the goal is to construct a graph in the target system that results in similar geometry while preserving as much parametric information as possible. In [Rappoport03] we have given an overview of our UPR solution to the FBDE problem. The STEP parametrics group has some open proposals for FBDE (parts 108 and 55).

FBDE retains design intelligence, allows modifications at the receiving side, and potentially avoids the geometric tolerances problem. On the other hand, it is not always technically possible to successfully exchange every feature (see the next section for an explanation of how our architecture handles this issue). Geometric DE usually works; however, when stitched solids are desired then success rates drop, modifications are very difficult to do, and design intelligence is lost. It is thus desirable to integrate both methods in the same system.

What does such an integration mean? The whole discussion in this paper is at the single part level, which is where the real challenge lies. Integration at the assembly level is an architectural, workflow issue. At the part level, integration means that every feature (or every feature sub-tree or sub-list) could be transferred to the target system either as features (parametrically) or as plain geometry (non-parametrically). In Section 3 we will give a formal problem statement.

In this paper we present a general concept, Geometry Per Feature (GPF), that achieves such an integration, along with algorithms

† Proficiency Inc.
‡ The Hebrew University of Jerusalem and Proficiency Ltd.

for implementing it. First we give a general background of feature based design and of our UPR FBDE architecture. We then give a precise problem statement, present three solution alternatives: delta solids (Section 5), delta boundary (Section 6) and delta faces (Section 7), and show that under normal CAD circumstances the latter is preferable. Implementation is discussed in Section 8.

2. Feature-based data exchange

In this section we give a brief background on the feature based design paradigm and of our UPR feature based data exchange (FBDE) architecture [Rappoport03].

2.1. General background

In the feature based design paradigm, the model is represented as a graph (or tree or list) of operations called features. The tree is sometimes called the 'history tree'. Operations create new geometry or modify existing geometry. Feature based design is basically an extension of constructive solid geometry (CSG). The differences are that in FB design there are more operation types; FB provides associativity to parameter changes through persistent naming of boundary entities [Kripac97; Rappoport97]; FB design heavily relies on implicit constraints (usually 2-D constraints and dimensions in sketches); and FB design includes surface operations and objects, and is not limited to solid objects. The terms 'feature' and 'operation' will be used in this paper interchangeably.

Although the main point with the FB paradigm is that operations are parametric, FB systems also provide non-parametric operations. In our context, a non-parametric operation is an operation that introduces into the model a piece of fixed geometry defined independently of the current state of the model, and potentially uses it in order to modify the model. Examples for non-parametric operations include the orphan and patch operations detailed in the following sections.

The goal in FBDE is to create a target model that is feature based as well, using features that are as similar as possible to the original, while keeping the geometry as similar as possible. Note that in general the geometry cannot be identical due to different tolerancing policies. As long as the approximation is controlled, this is totally acceptable in practice.

As explained in [Rappoport03], FBDE is difficult due to several reasons: inherent functional incompatibilities between CAD systems; feature semantics can in many cases be known at runtime only; and implementational incompatibilities between the CAD systems. Any solution to the FBDE problem must: (i) design for the case of a system not supporting a data item (operation) that is explicitly supported by another system, (ii) design for the case of incompatibilities and failures that can be discovered only by examining the run-time behavior of the systems involved, and (iii) be practically feasible.

2.2. The UPR architecture

The only complete FBDE solution described so far is the one in [Rappoport03]. Our *Universal Product Representation (UPR)* architecture is a star architecture, with data stored in a central representation called the UPR. Export and import modules create and read data to and from the UPR. Each feature has a unified data section and a set of *rewrites*. A rewrite is a different way of importing the feature that creates equivalent geometry. A rewrite may be a function only or a function with internal data.

Both the unified data and all the rewrites also store *verification data*, in order to identify success or failure of import. Usually, rewrites are invoked when import success level does not match the desired geometric quality.

The import module flow proceeds stepwise, adding one feature at a time. This is in general the only alternative, because adding a feature requires attaching it to the existing solid, and this attachment information may be dependent upon all preceeding features. When selecting a method for importing a specific feature, the factor that influences the decision the most is the set of operations available to the user in the target CAD system. After all, in FBDE the goal is to transfer an operation to an operation or a set of operations. That is, a source operation is emulated by one or more target operations. It is crucial to understand the operation repertoire of the target system in order to select the possible emulations and sort them according to perceived model quality. The availability of target operations or lack thereof are central for the discussion in this paper.

3. Integrated feature-based and geometric DE: problem statement

In this section we provide a formal problem statement and describe motivating usage scenarios and their implications on the problem statement.

3.1. Initial problem statement and notations

The problem we are dealing with can be phrased using the terminology of the previous section as follows: implement a rewrite of a feature F such that the resulting target model will be geometrically equivalent to the original (as much as possible), but will be purely geometric and will not contain any parametric, feature based information on feature F. We refer to any solution to this problem as a Geometry Per Feature (GPF) technique.

The above statement treats GPF as an integration of features and geometry at the single feature level. It would be a plus if the solution would be applicable with no substantial modifications to integration at the multiple feature level, that of a full feature sub-tree or sub-list.

Architecturally, a GPF solution may in principle be implemented at the import module only, but we allow it to require changes to both the export and the import modules. The solutions discussed in this paper all require changes to both modules.

Denote by G a parametric model in a source CAD system and by H the same model after adding a single new feature F. Assume that a FBDE application has already imported G into a target CAD system to create a model G', whose geometry is equivalent to that of G. We now seek a set of non- parametric operations $F_1, ., F_k$ that, when applied to G', will result in a model H' whose geometry is equivalent to that of H.

The conceptual difference between one GPF solution and another will lie in the nature of the non-parametric operations used at the target system.

It should be stated that we are mostly interested in providing a GPF solution for solids G and H that arise in real world situations. We would not mind that a GPF solution would pose limitations on the geometry and/or topology of G or H that are not significant in practice.

3.2. Usage scenarios

Once a GPF rewrite is available it can be used in different situations. There are two major ones:

1. Incidental: during FBDE, when a feature's import and none of its parametric rewrites match the desired target quality, GPF can be used as the next fallback (rewrite) to attempt.
2. Intentional: when it is desired to explicitly remove the parametric feature information from the target model, e.g., for business reasons.

Our problem statement requires that any solution will have to be practically implementable in the context of these two usage scenarios. The second is supported by the definition of what constitutes a GPF. Supporting the first one has three major implications as described below. First, it must be practically possible to provide a GPF rewrite for every feature in the source model. This means that preparation of the data needed for a GPF rewrite, either during export or during import, should not demand an excessive amount of storage and/or time.

The second implication relates to the capability of performing parametric changes on the target model, which is one of the main reasons why users want a feaure based DE rather than a purely geometric DE. In the incidental scenario, we would like to enable the user to modify as many of the parameters of G' features (features preceeding the feature F rewritten using GPF) as possible. The reader may incorrectly think that this is impossible with GPF anyway because GPF is by definition non-parametric. For example, parts are usually begun with an extrude feature based on a 2-D sketch; it is difficult to think how to implement a GPF of a further feature in the history tree in a way that will enable modifications of the major dimensions of that sketch. However, it should be realized that features rarely affect all of the geometry of the prior model. Rather, they are usually local in nature. A GPF of a feature usually leaves much room for parameter modification of previous features. For example, imagine a GPF for a round feature (even a round feature that affects most of the model's edges); it should not block the modification of the depth of a hole feature that is defined in the middle of a face and is not affected by the round feature at all.

The third implication relates to the cases when users need a 100% parametric model at the target system. In those cases GPF is not desired; it is present at the target model only because the FBDE software could not do better. However, it may still be possible for a human user to do what the FBDE software could not. Hence for such users GPF needs to be implemented in a way that enables them to replace the GPF operations by equivalent operations interactively.

In this paper we focus on the GPF technique by itself and not on its possible applications. Hence our problem statement does not cover any further applications of GPF beyond the two above, which already provide enough motivation for dealing with the problem. We will briefly touch upon one such application in the last section.

4. Import of full geometry

The simplest GPF method is import of full geometry. All CAD systems support an operation that creates new geometry given a Brep (of some sort) of that geometry. For example, in I-DEAS this operation is called 'orphan'. A simple GPF approach is to import the full geometry of the target model H as a single unit by using such an orphan operation. Most CAD systems require that the imported geometry be stitched to a solid if further solid features are to be defined on top of it.

Import of full geometry is an excellent GPF method for operations that create new geometry independently of existing geometry. Those are typically features at the leaves of the feature tree. However, it is not an attractive method for a general GPF.

With import of full geometry, the model G' that is present in the target system prior to the invocation of the orphan feature must be discarded somehow. If the feature is not a leaf of its feature tree, the pointset of G' is usually not the empty set. To erase the prior model, If the CAD system has an explicit operation that does this then it could be used (an example is placing the feature history in a special blank or no-show layer.) Otherwise its effect must be emulated by a feature combination. For example, one can create a box B completely containing G', then add a Boolean subtraction operation $G' - B$, thus transforming the represented pointset to be the empty set.

This solution is a valid rewrite, but it is not the most attractive one. There are three major problems with it:

1. Recall that in the incidental usage scenario of GPF we want to enable the user to manually replace the GPF operations by parametric ones having the same effect. The sequence 'erase model; import orphan' is very difficult to edit in such a manner, because the system does not represent in any explicit way the effect of the particular feature to be edited. This stands in contrast to the other GPF approaches described in the following sections.
2. This method requires the full geometry of H. A naive implementation would be to export it after each and every feature operation in the source system. This approach is obviously wasteful in storage and probably in computation time. A more sophisticated approach would be to compute and export only the changes that occurred in the original model geometry, and reconstruct the full geometry before importing it. We will not explain here the details of how to do this, because this is exactly the type of method discussed in the next sections of this paper, where reconstruction is done using the capabilities of the target CAD system.
3. Any associativity with the prior model G' is lost. By that we mean that parameters of G' features cannot be modified such that the result will be manifested at the target model H'. This contradicts one of our requirements as discussed in the previous section .

5. Delta solids

A second GPF method is based on solid differences, or delta solids. The idea utilizes the symmetric set difference between H and G, and the formula $H = G + (H - G) - (G - H)$. We compute the pointset that is added to G as a result of the feature F, $DA = H - G$, and the pointset that is subtracted from G as a result of F, $DS = G - H$. These two pointsets are transferred to the target system (using

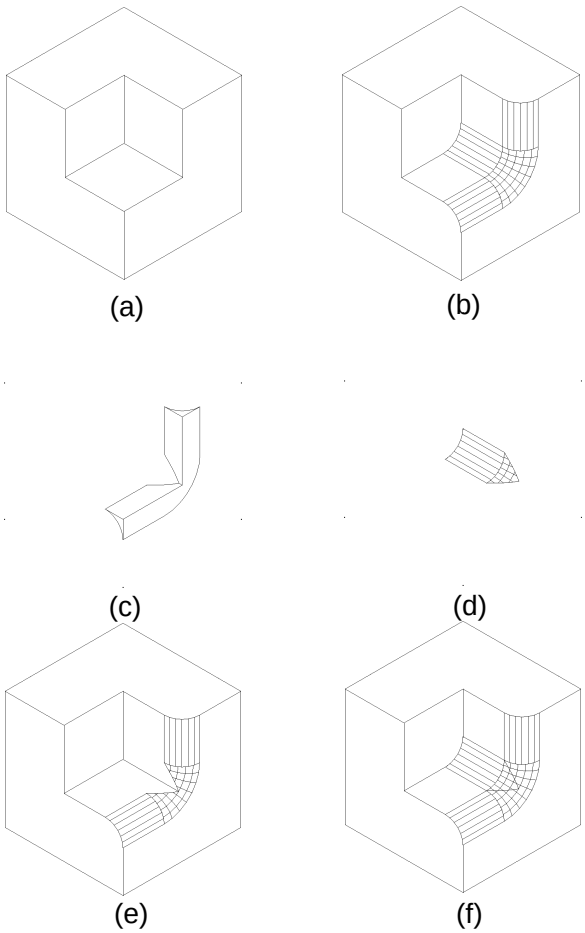

Figure 1: *Delta solids: (a) the model G; (b) the model H resulting from an application of a round feature; (c) the subtracted solid DS; (d) the added solid DA; (e) $G-DS$; (f) $G-DS+DA$.*

an orphan operation) to create the two solids DA' and DS'. Using these latter two pointsets, we then create H' using two Boolean operations, union and subtraction: $H' = G' + DA' - DS'$. See Figure 1.

In many cases due to the semantics of the feature F we can know in advance that one of these sets will be empty, and optimize the computation accordingly. Note that it is indeed possible for a single feature to result in both sets being non-empty, e.g, the round feature in Fig. 1 defined over convex and concave edges simultaneously. It is also possible for both sets to be empty. This is a rare, but interesting case. For example, some features modify the topology, but not the geometry. Recall that our only requirement was to reconstruct the geometry, hence the result will be valid. Note that in general, two CAD systems will represent the same solid using different topological representations.

Delta solids avoid the three problems with import of full geometry: the effect of the feature is represented in a localized manner at the target system; intermediate storage is optimized; and associativity in areas not affected by the feature is retained to a maximal degree.

However, the computations involved with a direct implementation of the delta solids method are both heavy and unstable. In general, Boolean operations are among the heaviest operations in solid modeling in terms of computing time. Moreover, in our specific case most of the boundaries of the participating solids overlap each other, which makes the operation very difficult to implement in a robust manner. In such a case symbolic methods, such as persistent naming, must be integrated with the general purpose Boolean operation procedure, notifying it of known overlaps. It is possible that such methods are used by CAD systems internally (CAD systems are aware of face and edge overlaps between feature operations due to their persistent naming mechanisms), but it is not guaranteed. This means that the FBDE system should probably implement robust Boolean operations by itself and not rely on the capabilities of the CAD systems. The delta faces method we will present below implements such a symbolic approach in a more elegant and efficient manner.

6. Delta boundaries

In this section we describe a boundary based alternative to delta solids, called delta boundaries. Delta boundaries forms the basis of delta faces, which is our optimized solution presented in the next section.

6.1. General idea

The boundary differences or delta boundaries method is similar to the delta solids method, but handles the boundary of the model instead of its solid pointset. The parts of the boundary of G that are removed by the new feature F are cut out; new parts added are glued in. One way to create H' is as follows. First, we compute $BDA = Boundary(H) - Boundary(G)$ and $BDS = Boundary(G) - Boundary(H)$. *BDA* and *BDS* are in general open surface sheets, not solids. We transfer *BDA* and *BDS* to the target system to create BDA' and BDS'. We now have to cut *BDS* off and glue *BDA* in, which in principle can be done in two different ways: first glue then cut, or first cut then glue.

The first option assumes that the CAD system supports the representation of non-manifold solids. In this case,

$H' = SubSheetRemove(SheetAdd(G', BDA'), BDS')$.

The semantics of the two operations follows the selective geometric complex (SGC) framework [Rossignac88]. $SheetAdd(P,Q)$ adds a surface sheet Q to the SGC entities of the a solid P. Crucially, even if the input solid P is manifold the resulting object will be a non-manifold solid, unless the sheet Q lies completely on P's boundary, which usually does not happen in our case. $SubSheetRemove(P,Q)$ is an operation that removes a subset Q of the SGC entities of a solid P. In our case, due to the special nature of the P and Q the result should actually be a manifold solid H'.

In the second option, $H' = CloseToSolid(X)$ where $X = SubSheetCut(G', BDS'), BDA')$. $SubSheetCut(P,Q)$ removes a subset Q of the boundary of a solid P, potentially transforming it into an non-solid surface sheet. $CloseToSolid(P,Q)$ adds a surface sheet Q to a surface sheet P and verifies that the resulting surface sheet defines a closed volume that can be regarded as a solid.

The delta boundaries method is more stable than the delta solids method, because the Boolean operations required for computing the delta surface sheets are more stable than Boolean operations

between solids (for example, faces that do not intersect can be simply neglected).

The problem with the two conceptual options described above is a very pragmatic one: the required operations are not readily available in most CAD systems. Most CAD systems do not support non-manifold solids at all, requiring their models to be either manifold solids or surface sheets.

6.2. Implementation using the patch operation

Most CAD systems do provide an operation, patch (or sew) that explicitly replaces sub-parts of a model's boundary by a given surface sheet, where both the input and the output model are manifold solids. Formally, the operation $Patch(P, Q)$ receives a solid P and an open directed surface sheet Q (that is, Q is a sheet having a material side) having boundary edges that are assumed to lie on the boundary of P. It glues Q to the boundary of P, and discards the portion of P's boundary that does not belong to the resulting solid. The material side information is crucial in order to determine which faces are those to be removed; the surface sheet boundary must partition the solid boundary into disjoint regions that can be oriented consistently with the material side of the surface sheet. In addition, the interior of the surface sheet must not intersect the boundary of the solid, except at the regions that are to be discarded by the patch operation.

Figure 2 shows an example. In (a) we see in full lines the model G' before invocation of the patch operation. The argument to the operation, *BDA*, is shown in dashed lines. We can still see *BDS* as well. In (b) we see the model H' after the patch operation. *BDS* has been removed, effectively replaced by *BDA*. *BDS* in this case is exactly the part of the boundary 'covered' by *BDA* in (a).

In some CAD systems, the patching surface sheet must be connected. In such systems we can apply the patch operation for each connectivity component separately. This technique will fail in the rare cases where both surfaces must be patched simultaneously (e.g., when a portion of a torus is cut off by two disks, a situation that will not arise in practice).

In some CAD systems, a patch operation is not provided directly, but it can be emulated using a combination of other operations. For example, I-DEAS provides operations that can be used to emulate patch but which require the stitching edges to be already present in the model. In this case those new edges should be created using Euler-like operations before the application of the stitch operation. This is a simple and straightforward technique.

BDS is not explicitly provided to the patch operation. As a result, it must be implicitly well defined by *BDA* and a direction. The patch operation cannot be used to implement the delta boundaries method when this condition does not hold. The condition does hold in almost all practical situations. One situation in which it does not hold is when the solid G has multiple connectivity components and one of them is completely deleted by the feature F. In this case, all of the faces of that component belong to *BDS*, but this piece of information is impossible to reconstruct by any *BDA* (e.g., *BDA* can be empty.) A similar situation occurs when the solid G has cavities and one of the cavities is completely removed by F.

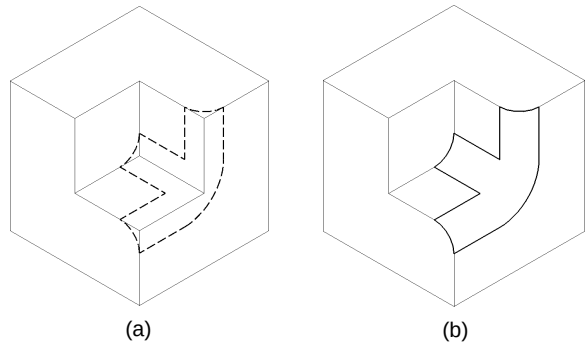

Figure 2: *Delta boundaries and delta faces: (a) before the patch operation; BDA is shown dashed; (b) after the patch operation.*

7. Delta faces

The delta faces method is an optimized variant of the delta boundaries method. It uses the idea behind delta boundaries and the fact that the patch operation is used when importing the GPF into the target system. The difference lies in the nature of the patched surface sheet: in the delta faces method, its computation can be done by purely symbolic data structure manipulation operations; no geometric computations are necessary.

7.1. Import: the patch operation

Import in the delta faces method is done exactly as in the delta boundaries method, using the patch operation. The surface sheet provided as input to the patch operation is different, as explained immediately below.

7.2. Export: a face cover of BDA

The delta faces method is based on the following observation. A face cover of *BDA* is a subset of the boundary of H that contains *BDA*. A face cover is a surface sheet. Any such face cover that is patched with G will produce H.

The minimal face cover of *BDA* is the minimal set of faces on the boundary of H that contains *BDA*. The immediate thinking is that it would be optimal to use a minimal face cover. The idea in the delta faces method is that a face cover that is reasonable (but not necessarily minimal) can be computed without geometric computations.

Computing a face cover is sometimes a simple matter. When the CAD system provides persistent naming for its boundary entities, then computing the delta faces is a matter of calculating the symmetric difference of two sets of face names. However, typically CAD systems do not provide access to persistent names. Nonetheless, all CAD systems provide information as to what new faces were created by a feature. What we need now are the identities of the faces that were modified by the feature.

Given the new faces information, we utilize the faces adjacent to the new faces as if they were the modified faces. This is formally incorrect in two cases. The first case is if the adjacent face was not modified. This does not pose any problem to the GPF export, because then the face cover is only larger than the minimal one, but would still produce correct results. The second case is the very rare

case of a face being modified without being adjacent to a new face. For example, a hole that is filled with material modifies the face that the hole was positioned on, but does not create any new faces in case the CAD system merges the former face with the new hole cover to a single face. Even in this case the results of the overall data exchange task would be correct, because our architecture includes feature verification information for every feature (see Section 2). If the model resulting after the application of GPF fails a verification test, this is treated just like any other feature failure: the execution is rolled back and a feature rewrite is invoked.

There is a limitation of the patch operation which may affect the delta faces method: if the face cover is closed, then it cannot be used with the patch operation. This is another rare case, which can occur with global operations, e.g., when all edges have been rounded. In these cases, the delta face is the full geometry as discussed in Section 5.

One must be careful as to what constitutes a new face and what constitutes a modified face. We assume that any change to a face (topological) or to its carrier (geometric) implies that the face has been modified, but faces may flip direction (material side) as a result of applying the feature. We call these inverted faces, and treat inverted faces as modified faces. Again, in all of these cases the worst that can happen is addition of faces to the delta faces set that are not strictly necessary but which cause no error to the result.

Computing the minimal face cover involves geometric computations, but can be computed as follows if desired. Start with the set of faces that were created by the feature. Obviously, these faces must be included in the covering. Next, add adjacent faces as needed. An adjacent face is needed if it bounds the created face on an edge that does not lie on the boundary of G. This is a necessary, but not sufficient, condition for minimal face cover faces, because of the very rare case of a face being modified without being adjacent to a new face.

8. Implementation

GPF has been implemented in a commercial product, the Proficiency Collaboration Gateway (CG). CG currently supports feature based data exchange between today's five high end CAD systems: Catia V4, Catia V5, ProEngineer, Unigraphics and I-DEAS. CG supports several versions of each system. The GPF implementation is stably integrated within CG since mid 2002. It has been successfully used in hundreds of thousands of DE operations exchanging real product data ranging from simple parts to huge assemblies.

The recommended mode of operation is to export a GPF for every feature in order that the GPF data be available in the UPR in case it is needed during import. This enables GPF import to be independent of the source CAD system. Another option is 'GPF on export failure', where GPF is computed in case a feature's export has failed for some unpredictable reason.

Figures 3–6 show a real world example of an angle bracket transferred from Unigraphics to Catia V5, ProEngineer and Catia V4. The resulting models are completely feature-based apart from a single 'complex hole' Unigraphics feature, which has been intentionally transferred as GPF. When the feature is not marked as to be transferred using GPF, the system transfers it as a fully parametric feature like all the other features in the model.

9. Discussion

We have raised the question of how to integrate geometric and feature based CAD data exchange, and presented several methods to solve it. Among them, the delta-faces GPF was shown to be the most attractive for practical usage due to its reliance on purely symbolic operations during export and on relatively stable geometric operations on import. An implementation which continuously satisfies real world customers has been done. In this implementation, GPF is mostly used as the ultimate single-feature rewrite in the UPR architecture.

Another advantage of GPF is that its rewrite data can be used by applications other than DE. For example, we have implemented a feature based viewer application that uses the faces exported by the GPF mechanism by highlighting them, thus providing a visualization of the geometric semantics of the feature. This is the first feature-based CAD viewer that is CAD system independent.

The discussion in this paper was done in terms of GPF for a single feature. Note that it is a simple matter to use the same techniques in order to provide a GPF rewrite for several features at once (e.g., a complete sub-tree or sub- list of the feature history). We simply take the union of the geometric entities created or affected by those features.

Since GPF is by definition an integration of geometric DE within a FBDE environment, it suffers from the usual problems of geometric DE. In particular, GPF can result in unstitched solids, which may pose a problem for the continuation of the FBDE task. In such cases, more radical rewrites may be needed (e.g., a global or semi-global geometric rewrite). If desired, the unstitched GPF can be stored in the target model as a detached set of surfaces, in order to help manual manipulation of the model. In our implementation, such problems rarely occur.

Acknowledgement. Michal Etzion actively contributed to the Proficiency GPF design and implementation. The content of this paper is patent pending.

References

[Bloor95] BLOOR M.S., OWEN J., Product Data Exchange, UCL Press, University College London, Gower Street, London, 1995.

[Hoffmann93b] HOFFMANN C.M., On the semantics of generative geometry representations. 19th ASME Design Conference, Albuquerque, New Mexico, September 1993.

[Kripac97] KRIPAC J., A mechanism for persistently naming topological entities in history-based parametric solid models. Computer-Aided Design 29(2):113–122, 1997.

[Rappoport97] RAPPOPORT A., The Generic Geometric Complex (GGC): a modeling scheme for families of decomposed pointsets. Solid Modeling '97, ACM Press, pp. 19–30.

[Rappoport03] RAPPOPORT A., An architecture for universal CAD data exchange. Solid Modeling '03, ACM Press, pp. 266–269.

[Rossignac88] ROSSIGNAC J.R., O'CONNOR M.A., SGC: a dimension-independent model for pointsets with internal structures and incomplete boundaries. In: Wozny, M., Turner, J., Preiss, K. (eds), Geometric Modeling for Product Engineering, North-Holland, 1988.

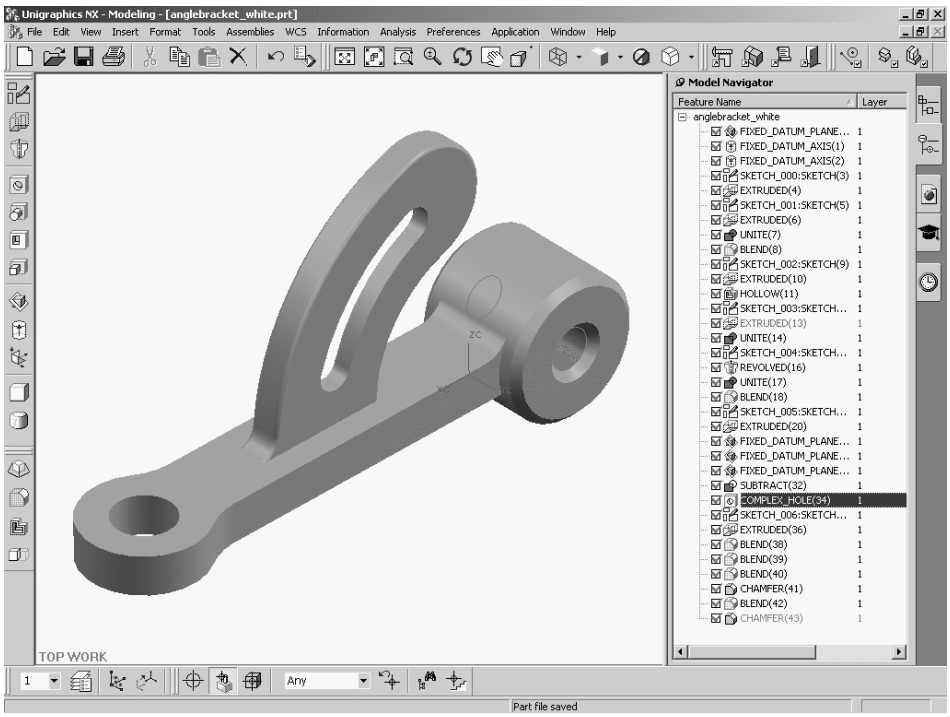

Figure 3: *The original angle bracket in Unigraphics. Note the highlighted 'complex hole' feature.*

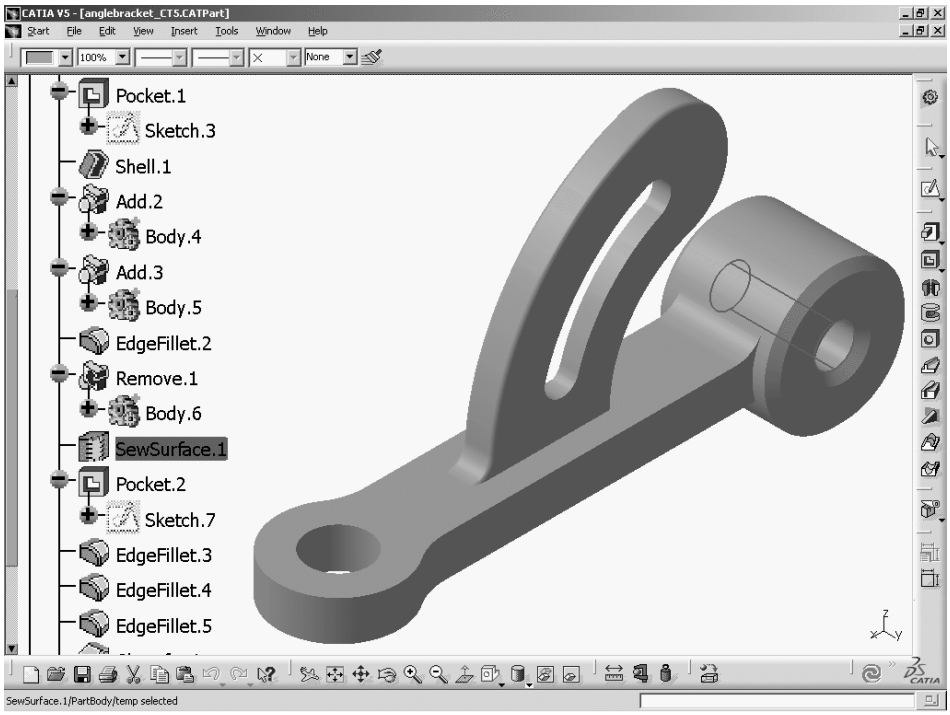

Figure 4: *Feature-based import of angle bracket into Catia V5 with a single GPF. Note the highlighted 'SewSurface' feature.*

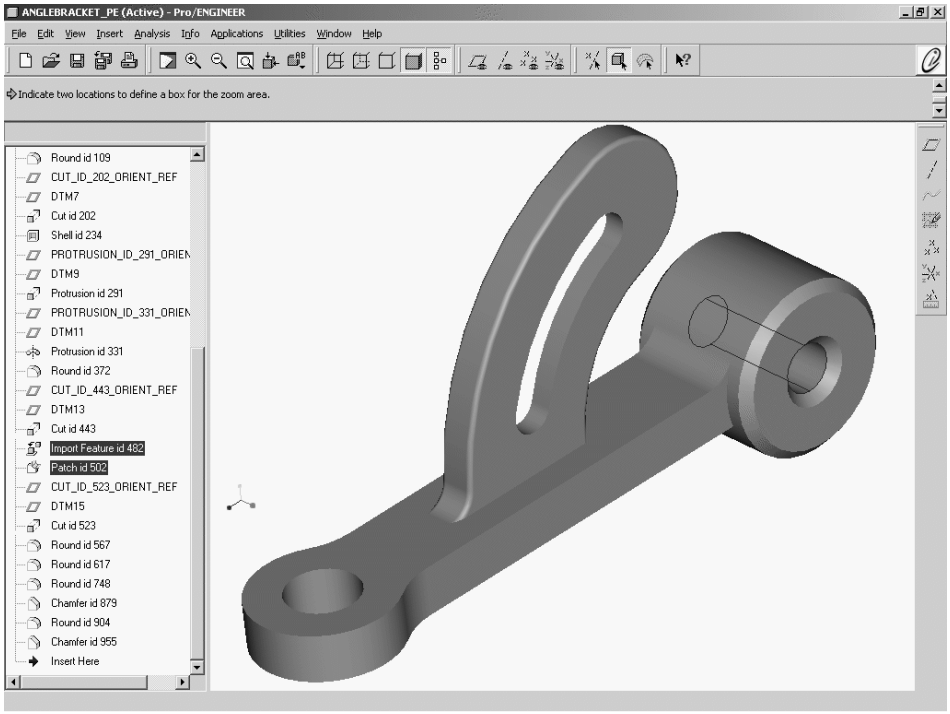

Figure 5: *Feature-based import of angle bracket into ProEngineer with the same GPF. Note the highlighted 'patch' feature.*

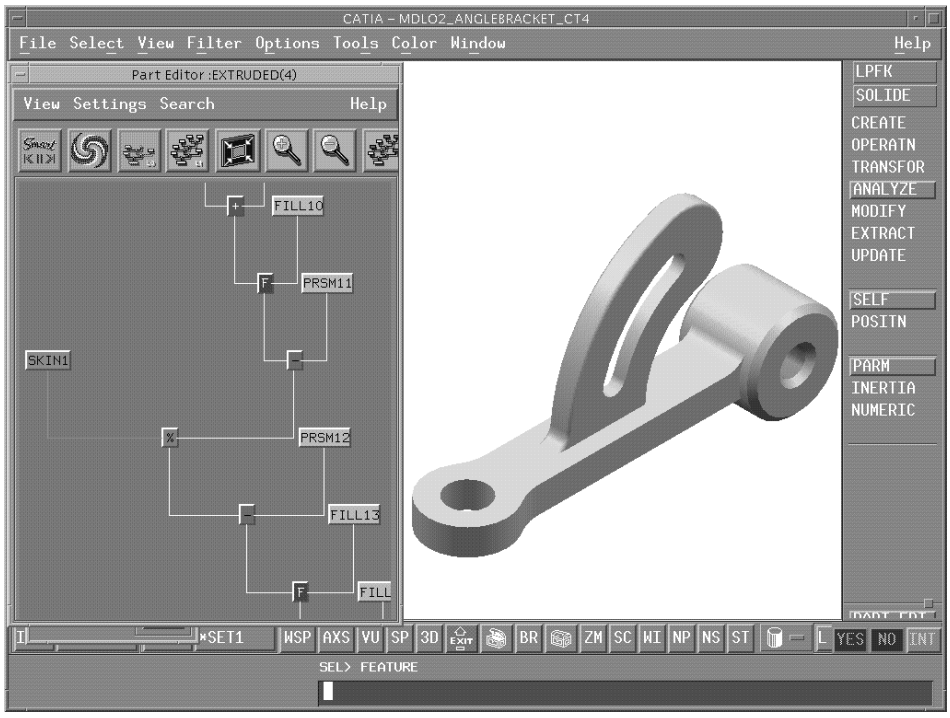

Figure 6: *Feature-based import of angle bracket into Catia V4 with the same GPF. Note the 'skin' feature on the left.*