

Project Integration Architecture: A Practical Demonstration of Information Propagation

Dr. William Henry Jones

National Aeronautics and Space Administration
John H. Glenn Research Center at Lewis Field
Cleveland, OH 44135
216-433-5862

William.H.Jones@grc.nasa.gov

X00.00 19 Nov 2001

X00.01 12 Mar 2002

Keywords:

PIA; PRICE; Knowledge Management; Application Integration; Information Propagation;
Technical Integration Technologies; Semantic Infusion; Self-Revelation;
Computer Aided Design; Inlet Analysis; Mesh Generation;
CAPRI; ProEngineer; Rocket-Based Combined Cycle Engine;

ABSTRACT: *One of the goals of the Project Integration Architecture (PIA) effort is to provide the ability to propagate information between disparate applications. With this ability, applications may then be formed into an application graph constituting a super-application. Such a super-application would then provide all of the analysis appropriate to a given technical system. This paper reports on a small demonstration of this concept in which a Computer Aided Design (CAD) application was connected to an inlet analysis code and geometry information automatically propagated from one to the other. The majority of the work reported involved not the technology of information propagation, but rather the conversion of propagated information into a form usable by the receiving application.*

1 Introduction

1.1 History

In the late 1980s, the Integrated CFD and Experiments (ICE) project [1, 2] was carried out with the goal of providing a single, graphical user interface (GUI) and data management environment for a variety of CFD codes and related experimental data. The intent of the ICE project was to ease the difficulties of interacting with and intermingling these disparate information sources. The project was a success on a research basis; however, on review it was deemed inappropriate, due to various technical limitations, to advance the effort beyond the successes achieved.

A re-engineering of the project was initiated in 1996 [3]. The effort was first renamed the Portable, Redesigned Integrated CFD and Experiments (PRICE) project and then, as the wide applicability of the concepts came to be appreciated, the Project Integration Architecture (PIA) project. The provision of a GUI as a project product was eliminated and attention was focused upon the application wrapping and integration architecture. During the intervening years, work has proceeded and an operational demonstration of

the PIA project in a C++, single-machine implementation has been achieved.

1.2 Key Contributions

The PIA technology provides a number of benefits. Among the more significant are the following.

1. Complete engineering process capture is possible to the extent desired.
 - (a) A complete derivational history of every project configuration investigated can be captured, producing an auditable trail from final design back to initial guess.
 - (b) Technologist's journals, notes, and the like can be captured, allowing the record of thinking to be retrievable in the context of the hard data of the project.
2. Integration of applications into a functional whole is possible, allowing for the complex analysis of entire systems.

3. Rigorous design configuration synchronization is enforced, eliminating mis-matched analyses between integrated applications.
4. The classic *n-squared* integration problem is solved through the use of semantically-defined parameters.
5. Dimensional unit confusion is eliminated by encapsulating in parameters a self-knowledge of their own dimensionality.
6. Quality values (good, bad, and, potentially, a range in between) are captured allowing bad data or designs to be retained in the record without concern that they might be inadvertently relied upon as being good.
7. Application integration is achieved without the necessity of re-coding those applications to the standard. The wrapping nature of the architecture decouples commitment to the integration standard from the capital assets of the wrapped applications.
8. The wrapping nature of the architecture also allows for multiple wrappers to the same application. Among other things, wrappers appropriate to the skill level of various users might be developed.

1.3 Demonstration of Information Propagation

As a part of the project effort, it was deemed necessary and appropriate to develop a working example of an application graph and the automatic propagation of information from one application to another. Because of the limited resources available to a research project of this kind, it was important to identify as simple a demonstration effort as possible, while still achieving a real-world result.

Two applications were selected for the demonstration: a Computer Aided Design (CAD) geometry information repository and an inlet analysis code. The information to be propagated was, of course, the geometry information of an inlet designed in the selected CAD system. The propagated information was then to provide the geometric input for an analysis of the inlet by the inlet analysis code.

This problem was directly drawn from the world at hand. The selected inlet was one under study at the Glenn Research Center for a Rocket Based Combined Cycle (RBCC) propulsion system. The inlet was interesting in that it was neither axisymmetric nor two-dimensional. Instead, it was an integrated bulge extending around one third of a circular cross-section fuselage, three such engines surrounding the entire vehicle. This gave the flow path a cross section something akin to an orange segment. This geometry had

proven difficult to handle with traditional tools. The geometry was particularly ill-suited to the input forms of the selected inlet analysis code.

The CAD program used to define the engine geometry was ProEngineer, a commercial product of PTC, Inc., in use at the Glenn Research Center. To avoid complete vendor dependence, the developed PIA wrapper incorporated the Computational Analysis PRogramming Interface (CAPRI) [4] technology developed under the auspices of other projects at the Glenn Research Center. CAPRI provides a single Application Programming Interface (API) that has been implemented in the context of a number of different CAD products. By switching between CAPRI implementation libraries, a consuming application may be made relatively insensitive to the actual CAD product originating the geometry information.

The inlet analysis was conducted by the Large Perturbation Inlet Analysis code (LAPIN), also separately developed under other efforts at the Glenn Research Center. This code is a one-dimensional, unsteady, time-accurate flow code used for the evaluation of inlet/flow control stability. The belief that LAPIN, being a one-dimensional code, was in any sense simple proved to be in error. In fact, a great many options for studying the response of the flow to various control actions (bleeds, bypasses, mass injections, and the like) are provided by LAPIN, making it a rather complicated code.

2 Implementation

The theory of information propagation within PIA has been previously reported [5]. The present demonstration has provided no alteration of that original theory.

2.1 The CAPRI/ProEngineer Wrapper

A PIA-compliant wrapper was generated to encapsulate geometry information obtained through CAPRI technology from CAD files generated by the ProEngineer commercial software product. The wrapper is fully documented on the central PIA web site and is the subject of a separate report [6].

The wrapper is, from an external viewpoint, unremarkable; however, the internal structure necessary to implement the wrapper is of some interest. The key consideration dictating the resulting structure is the fact that the ProEngineer software product provides access to its geometry kernel only through the mechanism of a Dynamic Link Library (DLL) containing the consuming code. The DLL is identi-

fied to the ProEngineer executable image at program load time. The ProEngineer executable links to the identified DLL and executes the well-known entry point provided by that DLL.

In order to use some of the dynamic CAPRI API features (as will be discussed shortly), the wrapper must spawn the ProEngineer executable as a separate process and communicate with the wrapper-supplied DLL identified to that process as a backend, geometry server. Thus, while geometry information presented by the wrapper appears to originate in that wrapper, it is in fact produced in a separate process and communicated to the wrapper for presentation.

2.2 The LAPIN Wrapper

A PIA-compliant wrapper was generated for the LAPIN code. The wrapper is, again, fully documented on the central PIA web site; however, due to its conventional nature, it is not otherwise reported at this time.

It should be noted that the LAPIN code is maintained as a separate entity from the wrapper. When execution of the code is needed, the wrapper writes the Fortran namelist text file expected by the LAPIN code, executes the code, and then reads the text output file generated by that operation. Thus, the PIA-compliant wrapper requires no modification of the LAPIN code for its integration into the PIA application environment.

2.3 Geometry Products

One of the key technologies of the PIA information propagation formulation is the infusion of semantic meaning into parameter objects through the act of class derivation; that is, that a piece of consuming code can determine the kind of information being supplied by determining the kind of object supplying it. In the present demonstration, this technology allows the consuming LAPIN wrapper code to identify and obtain geometric assembly information by looking for parameter objects of the kind **PacParaGeoAsmb**. Having found such an object, the consuming code is then free not only to acquire the information the parameter object directly offers, but also to avail itself of the functional products provided by the object. In the case of the **PacParaGeoAsmb** geometric assembly parameter object, there is one principal functionality of interest to the consuming LAPIN wrapper code: the generation and manipulation of cross-sectional curves.

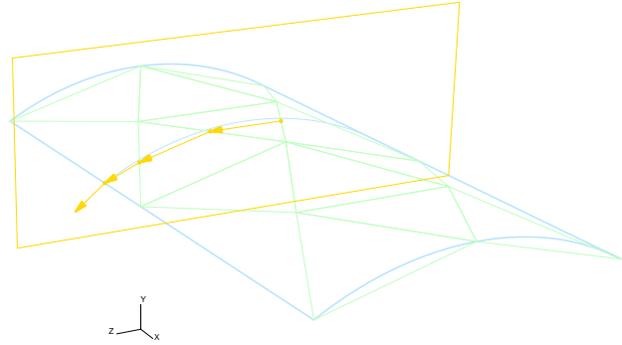


Figure 2.1: Triangle Intersection Tracking

2.3.1 Basic Generation of Cross-Sectional Curves

The **PacParaGeoAsmb** geometric assembly object provides several member functions which generate cross-sectional curves. These curves encompass the entire assembly encapsulated by the presenting object. Assemblies, though, imply a collection of things and, thus, do not directly provide the basis for cross-sectional curve generation.

In point of fact, assemblies organize boundaries and other assemblies, and boundaries, in turn, organize faces. In the CAPRI formulation (which is closely followed by the PIA wrapper), faces are made substantial through triangular tessellations; that is, a geometric face is ultimately realized as a set of points in geometric space that are organized into sets of connected triangles, the whole resulting in an approximation to the true geometric shape.

The tessellation information is the basis upon which cross-sectional curve generation begins. As shown in Figure 2.1 each triangle of a tessellation is examined until one is found which intersects the sectioning surface. Once such a triangle is found, the two intersections of the triangle's sides with the sectioning surface are computed. Curve generation then proceeds as an iterative process from these starting points by identifying the connecting triangle and computing the intersection of its side with the sectioning surface. Due attention is also provided to the possibility that a triangle vertex may lie exactly on the sectioning surface.

The geometric model does not provide closed faces; for example, what is in fact a cylinder is represented in the geometric model as two half-cylinders joined at their edges. Thus, the tracking of a cross-sectional curve around a closed solid must (and does) account for the crossing of edges shared between faces. While requiring the exploration of additional data structures, the fundamental logic of the curve generation process is the same.

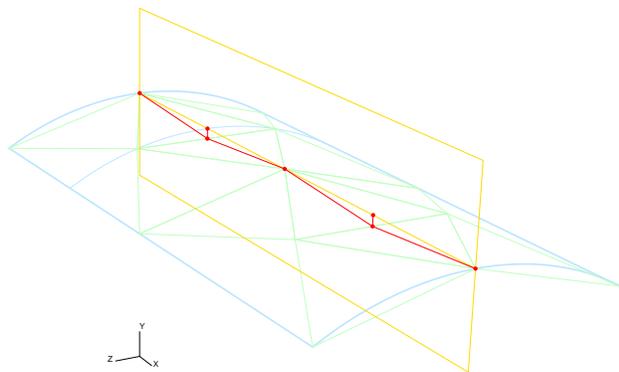


Figure 2.2: Introduction of Noise by Geometric Discretization

The cross-sectional curve generation process notes the starting point of its operations and, should it re-encounter that point during a subsequent iteration, it concludes that the produced curve is closed and makes an appropriate notation; otherwise, the curve defaults to an open status. Note, though, that since the supported CAD products, in this case ProEngineer, model solid objects, open cross-sectional curve results should not occur as a practical matter. It is possible, though, to have multiple closed curves result from a single cross-section, for example as the cross-section of a block with a hole through it.

2.3.2 Curve Fidelity Improvement

The cross-sectional curve generation process recognizes a fact of geometry discretization: when the geometric face is not flat, the collection of triangles is only an approximation to the shape of that actual geometric face. While it is true that each vertex of the set of tessellating triangles lies on the geometric face (to within an arbitrary value), it is not necessarily true that the line segment connecting each vertex pair lies on that face. Thus, a cross-sectional curve product based solely upon the intersection of triangle sides with a sectioning surface will introduce error into the portrayed geometry proportional to the curvature of the geometric face.

This introduction of geometric noise can be seen on close inspection in Figure 2.1. Figure 2.2 especially illustrates this by showing the axial cross section of a cylinder in comparison with the cross section produced by tracking the intersection of the tessellating-triangle sides (that curve being shown in red). When the triangle side spans the sectioning plane, the produced cross-sectional curve dips down to the height of the spanning side and deviates from the true geometric surface.

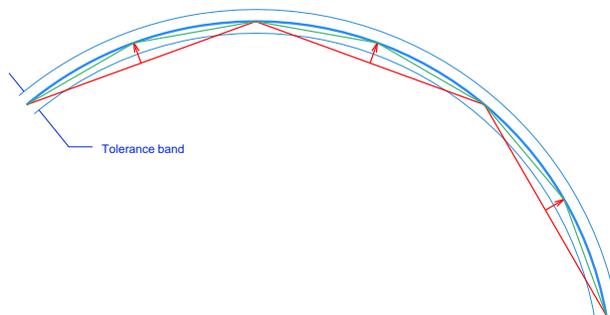


Figure 2.3: Curve Fidelity Improvement by Segment Bisection

The cross-sectional curve generation process removes this introduced error by utilizing a snap-to-face capability provided by the CAPRI API. As each tessellating triangle side is intersected with the sectioning surface, the computed point is then snapped onto the underlying geometric face. Because the direction of this movement to the face is unpredictable, the point improvement is, in fact, performed iteratively: the point is snapped to the underlying geometric face, then back to the intersecting surface. The iterative process terminates when meaningful movement of the improved point ceases.

The process to this point generates a cross-sectional curve whose defining discrete geometric points lie (to within an arbitrary accuracy) on the underlying geometric faces; however, just as with the sides of the tessellating triangles, the line segments implicitly joining the successive points of the curve do not necessarily lie in the underlying geometric faces.

To adjust for this curve-segment difficulty, a curve improvement phase is performed. As shown in Figure 2.3, each line segment of the curve is bisected and the resulting point snapped onto the geometric face, just as the original tessellation intersection points were. If the displacement of this point improvement is greater than the accuracy specified for the cross-sectional curve, then the improved bisection point is inserted into the curve and the two curve segments that result are, themselves, recursively considered for bisection point improvement.

2.3.3 Curve Purification

As noted earlier, assemblies usually organize two or more boundaries or other assemblies, each of which produces at least one cross-sectional curve (assuming, of course, that the sectioning surface intersected the organized geometric element at all). When boundaries touch each other across a portion of a face, as they will in many real-world geomet-

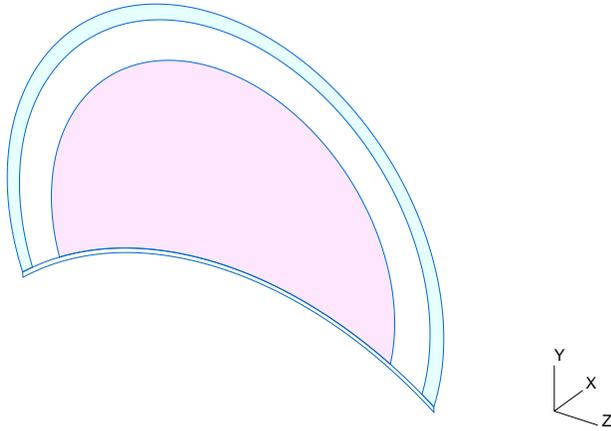


Figure 2.4: Cross Sections of Individually-Modelled Engine Components

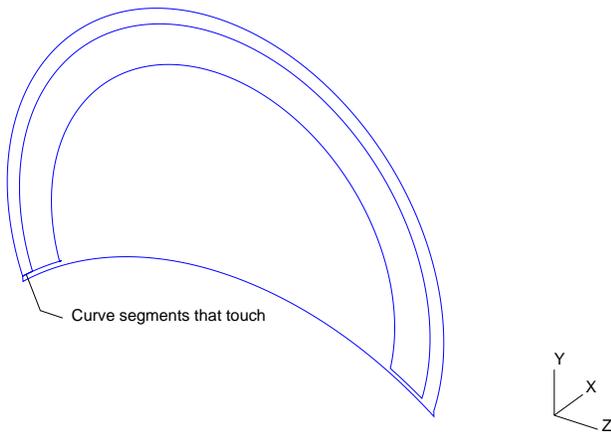


Figure 2.5: A Single Curve that Intersects Itself

ric assemblies, parts of the produced cross-sectional curves will be redundant. For example, consider the situation illustrated in Figure 2.4 which shows the lateral cross-section curves of a propulsion system in which the centerbody and cowl both sit on a common splitter plate. Portions of those cross-sectional curves obtained from both the centerbody and cowl duplicate portions of the splitter plate cross-sectional curve. The objects encapsulating cross-sectional-curve products provide functionality to identify and eliminate such intersections between curves, producing in such an event one cross-sectional curve where two (or more) existed before.

There is a further aspect to such curve merging. Consider an assembly of two objects in which the objects touch each other in two distinct areas, as the cowl touches the splitter plate in Figure 2.4. The resulting cross-sectional curves will intersect each other in two distinct places, namely at

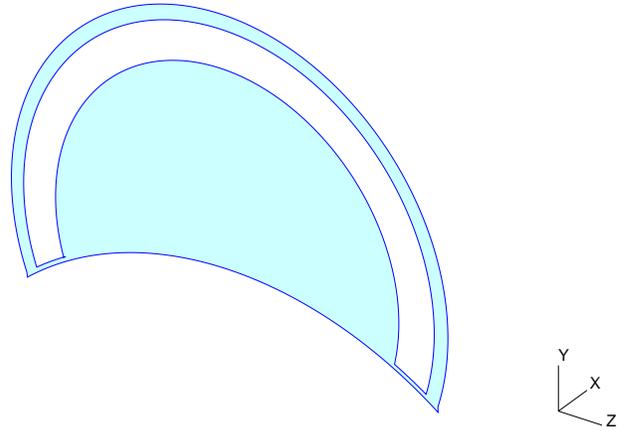


Figure 2.6: A Purified Curve Set that Reveals the Flow Path

each end of the arch. When curve purification detects one of these intersections and merges the two curves, the result will be a single curve that intersects itself at the other point of intersection, as shown in Figure 2.5. The implemented curve objects also provide functionality to detect this situation and split such single curves back into two non-intersecting (or pure) curves.

The cross-sectioning functionality of assembly objects utilizes these capacities to merge and split the set of curves obtained until a pure set of non-intersecting curves is obtained, as shown in Figure 2.6. In this way, curves that are cross-sections of the assembly as a whole are obtained for use by consuming code.

2.4 Information Consumption

The LAPIN analysis code requires geometric input that is, considering the fact that it is all the same propulsion system, radically different from that provided by the CAPRI/ProEngineer geometry wrapper. The most striking difference is that, while the CAPRI/ProEngineer information models the solid objects that make up the propulsion system, LAPIN focuses instead on the hole through space (that is, the flow path) left over by these solid objects. Furthermore, LAPIN is only interested in the profile of the flow path; the three-dimensional shape of the flow path is entirely beyond its scope of interest.

The LAPIN wrapper uses the cross-sectioning geometry functionality provided by the **PacParaGeoAsmb** geometric assembly object it identifies to transform the information encapsulated in the CAPRI/ProEngineer wrapper into a form useful to the LAPIN analysis code. Furthermore, by examination of the results it obtains, the wrapper is able to select among a number of geometry options accepted by

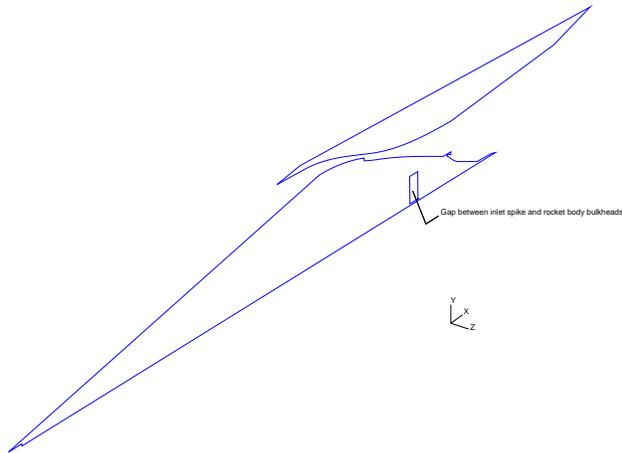


Figure 2.7: Propulsion System Flowpath Profile

the LAPIN code.

The following actions are performed to lead to this overall result during the act of information propagation.

1. The LAPIN wrapper obtains from the identified **Pac-ParaGeoAsmb** object a cross-sectional profile of the propulsion system, resulting in a set of curves like those depicted in Figure 2.7.

2. An attempt is made to reduce the curve set by eliminating those curves that exist in the interior of another curve.

This action is required to account for the actual situation of the RBCC geometry. The propulsion system consists of a cylindrical splitter plate, a rocket body mounted to the aft portion of that plate, an inlet spike mounted to the plate whose aft section fits over the rocket body and is able to translate longitudinally for shock capture, and a cowl bridging from edge to edge of the plate and generally enclosing the overall propulsion system. With the spike translated forward, the production of pure, profile cross-sectional curves, as shown in Figure 2.7, results in an apparent hole in the centerbody profile which is, in fact, the interior longitudinal gap between the inlet spike and the forward wall of the rocket body.

3. The profile curves are split at their longitudinal extremes to produce a set of open profile curves. The set is then vertically sorted.
4. The number of open profile curves is then used to select two profile curves for further processing.

- (a) If there are exactly four curves, the middle two curves are selected. This case presumes that one

of a number of centerbodyless flow path forms is represented by the geometric information.

- (b) If there are exactly six curves, the fourth and fifth curves are selected. This case presumes that a centerbody with upper and lower cowls is represented by the geometric information.

5. The two selected curves are examined to determine if they are, in fact, mirror images of each other. (This is another functional capability offered by the class of objects encapsulating cross-sectional curves.) If this is the case, curve processing is performed under the presumption that the flow path is of a centerbodyless, axisymmetric system, a type recognized and supported by LAPIN. Further discrimination of the geometry information is avoided.

6. The vertical coordinate of the leading point of the lower curve is compared to zero. If it is, essentially, zero, then curve processing is performed under the presumption that the flow path is of an axisymmetric system with a centerbody, another type recognized and supported by LAPIN. Further discrimination of the geometry information is avoided.

7. Should consideration of geometry information reach this point, all other relevant options presently implemented by LAPIN have been exhausted. The selected curves are processed under the presumption that the flow path is of a non-symmetric, two-dimensional system, a special type again recognized and supported by LAPIN.

In processing this last type, LAPIN accepts width geometry input in addition to its normal profile geometry. Pure, assembly cross-sectional curves normal to the longitudinal axis, such as those shown in Figure 2.6, are acquired from the assembly object and the second largest area (another function provided by curve objects) enclosed by a curve of the set is taken to be the flow path area. (The largest area is presumed to be the projected frontal area of the propulsion system, while smaller areas would be considered to be internal ducts for bypasses, bleeds, and the like.) A simple calculation based upon the vertical extent of the profile at that station then produces the width value for use by LAPIN.

It should be noted that, internally, LAPIN is concerned with the flow path profile and its cross-sectional area. The original LAPIN code made a universal axisymmetric presumption, allowing cross-sectional area to be computed directly from the centerbody (if present) and cowl profiles. The introduction of two-dimensional inlets to real propulsion systems necessitated the amendment of LAPIN to ac-

cept width input because the adjustment of profile values to achieve proper areas under an erroneous axisymmetric presumption would introduce errors into the oblique shock impingement computations performed by LAPIN.

The fact that the RBCC flow path is orange-segment shaped, rather than simply rectangular, is beyond the scope of LAPIN computations because no lateral effects exist in the LAPIN formulation. Although the actual flow path is an angular portion of an axisymmetric flow path (that is, the flow path is made from angular portions of surfaces of revolution), it was considered inappropriate to simply analyze a full axisymmetric extension since this would require the scaling of various bleed and bypass flow values.

3 Enhanced CAPRI/ProEngineer Services

The curve purification discussion identifies the activities necessary to deal with flow path geometry in an environment in which an engine is described as an assembly of separate pieces; cowl, centerbody, splitter plate and the like. The great majority of computation performed involves the integration of that separate information using only the tools of a discretized geometry; however, the ProEngineer product has the capability to eliminate this complexity by merging the separate pieces of the engine assembly into a single solid object. By dealing with the engine as a single solid, the need for curve purification is entirely eliminated and, with it, a very large computational effort.

A new release of the CAPRI libraries recently incorporated into the PIA project now includes the ability to invoke these assembly-merging facilities of ProEngineer. Performing this function has been made an optional part of the CAPRI/ProEngineer CAD wrapper.

It is worth mentioning that, due to the PIA technology, no reprogramming of the LAPIN wrapper was required to adapt to the revised CAPRI/ProEngineer wrapper since the LAPIN wrapper is programmed to the kinds of information it wants, not to the application presenting that information.

4 Results

The results, per se, are very simple: geometry information was presented by the CAPRI/ProEngineer CAD wrapper and consumed by the LAPIN wrapper during the act of information propagation. In and of itself, this result is rather anticlimatic. The same feat was accomplished using manual methods by the original engine analysis team, requiring several weeks of effort for each studied design.

A further element of the result, though, is the speed improvement obtained by automating the transfer of information. The runtime required when the actions of curve purification in a multi-solid, discretized geometry environment were required was on the order of five days, a figure far above the “instantly” commonly expected of such automation, but still an improvement over manual methods. Contrasted with this, the execution time when a pre-merged, single-solid formulation of the geometry was used was on the order of minutes, ranging somewhere between 12 and 30 minutes depending upon the granularity of the tessellation produced by the geometric services. The difference in these two numbers is virtually all in the effort of curve purification, which is eliminated by the single-solid formulation.

Perhaps what this runtime difference points out most forcibly is the debilitating effects of geometry discretization. While the reduction of parametric geometry (that is, the geometry of planes, cylinders, cones, and the like) to simply a maze of organized points may be unavoidable to achieve a least-common denominator between the many commercial CAD products, that loss of semantic information is huge step into the abyss. The ProEngineer multi-solid merging process is accomplished in a matter of seconds while the same result in the discretized reduction takes days. This points, once again, to that perpetual Holy Grail of the CAD world: a supported, common, high-level, open standard for geometric representation.

5 Documentation

Complete, class-by-class, member-by-member documentation is available on a central server the the Glenn Research Center. The root URL for this documentation is

<http://www.grc.nasa.gov/WWW/price000/index.html>

It must be strongly emphasized that these pages are the informal generation of the researchers involved and do not, in any way, shape, or form, represent an official statement of the Government of the United States.

References

- [1] The American Society of Mechanical Engineers. *Integrated CFD and Experiments Real-Time Data Acquisition Development*, number ASME 93-GT-97, 345 E. 47th St., New York, N.Y. 10017, May 1993. Pre-

sented at the International Gas Turbine and Aeroengine Congress and Exposition; Cincinnati, Ohio.

- [2] James Douglas Stegeman, Richard A. Blech, Theresa Louise Benyo, and William Henry Jones. Integrated CFD and Experiments (ICE): Project Summary. Technical memorandum NASA/TM-2001-210610, National Aeronautics and Space Administration, Lewis Research Center, 21000 Brookpark Road, Cleveland, OH 44135, December 2001.
- [3] William Henry Jones. Project Integration Architecture: Application Architecture. Draft paper available on central PIA web site, March 1999.
- [4] Robert Haimes. *Computational Analysis Programming Interface (CAPRI): A Solid Modeling Based Infrastructure for Engineering Analysis and Design*. Cambridge, MA, November 1999. Web Reference: <http://raphael.mit.edu/capri/>.
- [5] William Henry Jones. Project Integration Architecture: Inter-Application Propagation of Information. Draft paper available on central PIA web site, December 1999.
- [6] American Institute of Aeronautics and Astronautics. *Project Integration Architecture (PIA) and Computational Analysis Programming Interface (CAPRI) for Accessing Geometry Data from CAD Files*, number 2002-0750. Aerospace Sciences Meeting and Exhibit, Reno, NV.