

# Project Integration Architecture As a Foundation for Autonomous Solution Systems: The Postulation of a Meaningful “Solve Yourself” Method

*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 31 May 2002

## Keywords:

PIA; PRICE; CORBA; Knowledge Management; Network Computing; Application Integration;  
Technical Integration Technologies; Semantic Encapsulation; Collectives; Federations;  
Autonomous Solution Systems; Autonomous Logic; Reasoning Systems;

**ABSTRACT:** *The Project Integration Architecture (PIA) uses object-oriented technology to implement self-revelation and semantic infusion through class derivation. That is, the kind of an object can be discovered through program inquiry and the well-known, well-defined meaning of that object can be utilized as a result of that discovery. This technology has already been demonstrated by the PIA effort in its parameter object classes. It is proposed that, by building on this technology, an autonomous, automatic, goal-seeking, solution system may be devised.*

## 1 Executive Summary

It is proposed that the key object-oriented technologies of self-revelation and semantic infusion through class derivation combined and furthered by the Project Integration Architecture (PIA) effort may form the basis upon which an Autonomous Solution System (ASS) may be built. Such systems will automatically formulate approaches to posed problems from the available resources of a PIA collective and seek optimal solutions. Further, the automation of this process will enable the application of system analyses far beyond the levels attainable with current manual integration technologies.

The following benefits are expected to accrue from such technology.

1. The teaming process will be automated with an attendant increase in reliability.
2. An audit record of the solution process will be automatically generated.
3. Alternative solution strategies will be dispassionately

considered.

4. Risk assessment within a given solution approach will be automatically generated.
5. Automation of the teaming process will reduce team duties for discipline experts and make more time available for the advancement of those disciplines.
6. Integration technology will allow more-reliable, higher-fidelity analyses to be developed within narrowed disciplines.
7. Distributed server technology will reduce software maintenance and administration costs.

## 2 Introduction

### 2.1 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 computational fluid

dynamics (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 unavoidable technical limitations accepted at the time of project inception, to advance the effort beyond the successes achieved.

A re-engineering of the project was initiated in 1996 [3, 4, 5, 6, 7, 8, 9, 10, 11]. The effort was first renamed Portable, Redesigned Integrated CFD and Experiments (PRICE) and then, as the wide applicability of the concepts came to be appreciated, Project Integration Architecture (PIA). 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. This demonstration includes the integration of a Computer Aided Design (CAD) geometry-wrapping application with a wrapped CFD code and the automatic propagation of geometry information from one to the other [5]. Meanwhile, progress on a Common Object Request Broker Architecture (CORBA)-served, distributed-object implementation of the architecture is well underway.

### 3 Key Concepts

The PIA effort has capitalized upon two key concepts. Neither is original to this particular effort, but these concepts may have been combined and carried further than they have otherwise previously gone.

#### 3.1 Self-Revelation

Self-revelation is simply the ability of something to tell an inquirer what it is. This is an entirely common concept with people: one walks up to a person at a party and asks her profession, to which she replies doctor, or lawyer, or whatever. Further interaction is usefully directed from that self-revelation by the inquirer's knowledge of the identified profession.

In non-object oriented languages like Fortran or Cobol (at least in their traditional forms), the capability for self-revelation is somewhat difficult to devise; however, in object-oriented languages with class derivation and inheritance this becomes a relatively natural thing. Indeed, some language implementations provide the basic mechanisms

as a default, sometimes unavoidable, condition because of the conveniences it offers to development tools and the like.

#### 3.1.1 Self-Revelation of Kind

As demonstrated in the PIA foundation classes, it is an easy thing to declare in the patriarch of a class system the ability to respond to an inquiry with a string or a coded number that indicates the kind of object at hand. This is the very essence of self-revelation, a self-revelation of kind. In the PIA system, this self-revelation of kind is taken a bit further in that a concept of depth is supported, reporting for any given object of any derived class the derivational depth from the patriarch and the kind of the object at each derivational layer from the object's surface to its patriarch. (Such an examination is called, in the PIA nomenclature, an **ecdysiastical** analysis, from the Greek *ekdysis*, *ekdyein*, to get out of, strip off.)

#### 3.1.2 Self-Revelation of Content

Another slightly different form of self-revelation exists: a self-revelation of actual content rather than of kind. For example, the PIA Application Architecture defines that all application objects have, among other things, a set of operations that they perform; however, whether or not that set is empty and, if it is not empty, precisely what operations are in that set is left as a matter of further discovery, the discovery of content. A consumer of an application object is responsible for traversing the operations set and doing further analysis of the objects found there to determine what operations may be performed.

### 3.2 Semantic Infusion through Derivation

The semantics of something may be defined by encapsulating it in a class derived from another class. Usually, a patriarchal class is defined as a fundamentally amorphous blob with a few basic abilities. Derivatives of that patriarch are then defined with some basic statement of what objects of each derivative have and do. Additional layers of derivation are added, at each stage adding refinement as to the nature and function of the derivative class.

In the PIA effort, classes immediately derivative of the patriarchal class encapsulate the concepts of arrays, matrices, maps, strings and the like; however, the question of just exactly what it is an array or matrix or map of is left completely open. The next derivative layers begin to fill in those blanks; a map sorted by strings, for example. But still a map usually goes from something to something, and that question is not answered until the next derivative

layer in which it is discovered that it can be a map of strings to other strings, to integer values, to floating point values, or to objects, depending upon which derivative class is selected.

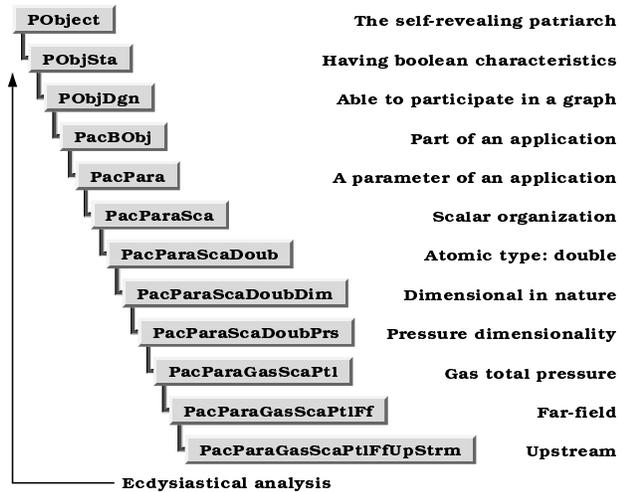


Figure 3.1: Semantic Infusion through Class Derivation

It is this pattern of semantic infusion through class derivation that is exploited by PIA. At some point (as illustrated in Figure 3.1), a derivative class becomes a parameter, whatever a parameter is. Beyond that, a parameter becomes a scalar, a vector, a matrix, an organization, and the like, then a scalar integer, a vector of floating point values, and so on. And beyond that, a parameter can become a dimensional parameter, and then a parameter whose dimensionality is of pressure or speed or length. Then a pressure parameter can become a gas pressure, then a total gas pressure, then a far-field total gas pressure, and then an upstream far-field total gas pressure, and so on until just exactly what the parameter is is utterly defined and revealed by the type of its encapsulating class.

This semantic infusion is not limited to parameters alone. It can be employed for operations, applications, indeed for anything whose characteristics can be identified and then further refined. Thus, an application can start out as just an application, then be refined to an analytical application, then to a fluids analysis application, then to a fluid flow analysis application, and so on.

### 3.3 The Combination of the Concepts

As is probably self-evident by now, the combination of self-revelation with semantic infusion through derivation is of enormous potential. It makes it possible for a piece of code to be handed a reference to an unknown object and discover what it is and, with appropriate need and knowledge

placed in that consuming code, put that object to use. For example, code can be written that discovers that an object is, in fact, an upstream, far-field Mach number parameter and, presuming that that is the kind of information the code wanted to have, that encapsulated number can be put to use.

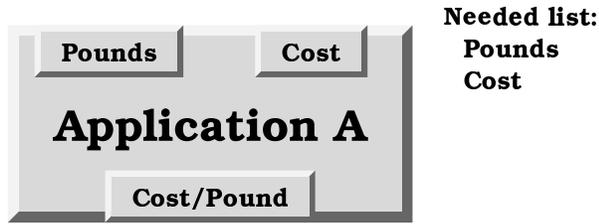
Further, through the self-revelation of content it can be discovered what other parameters are associated with that revealed Mach number. In the PIA formulation of applications, parameters exist in **configurations** which both hold other parameters and inherit parameters from their ancestral configurations. A search of these related parameters may reveal the altitude at which the Mach number exists, the total temperature, pressure, whatever.

The PIA formulation further indicates that configurations exist as components of applications, so self-revelation can be pursued to the point of discovering the kind of application in which this Mach number exists. Other mechanisms define whether the Mach number is an output of the application or an input to it. While not yet implemented, the ability to inquire as to the confidence that may be placed in this Mach number is also forseen: is the analysis of such high fidelity that the Mach number result it produces is very likely to be right, or is it just an estimation facility to get into the right ballpark?

All of this discussion presumes that the client code wanted Mach number information. If Mach numbers are irrelevant to the situation, for example because toasters are being designed, then the Mach number is easily ignored while information more relevant to the situation is awaited. In point of fact, client code need have no coded knowledge of Mach numbers at all. All that is needed is an escape route for “unrecognized”. Such a route is entirely sufficient for handling untold thousands of parameter forms.

## 4 The Postulate: A SolveYourself Method Can Be Devised

With this combination of self-revelation and semantic infusion in hand, it is now postulated that, given a sufficiently rich environment, there exists a set of problems for which a useful **SolveYourself** function can be devised. The key characteristic of this set of problems is that they must have a clearly defined answer. True love is not yet such a problem, dollars per useful pound to low earth orbit is much more likely to be so. Furthermore, problems for which a SolveYourself method can exist are often likely to be cast as optimization problems: get the best, get the most for the least. This is a problem formulation with wide applicability.



**Needed list:**  
Pounds  
Cost

Figure 4.1: An Automatically-Identified Application Producing a Desired Result

It is proposed that, given the PIA technology base and a sufficiently rich set of application wrapped in that technology, the solution of such problems can be dealt with in an automatic manner. A search of available parameter forms could reveal what kinds of parameters encapsulate the desired result and what available applications produce those parameters. For example, in a sufficiently rich PIA environment, a cost-per-useful-pound-to-orbit parameter would exist and there would be some application, illustrated in Figure 4.1, that computed that sort of parameter as an result of its operation. Automated examination of the possibilities would be able to identify these applications.

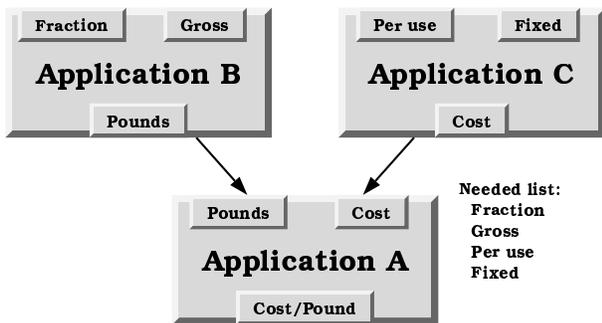


Figure 4.2: Automatically-Identified Applications Producing Needed Inputs

If an application can be identified as producing the desired result, the inputs required for that application can be identified and a recursive search to satisfy those needs begun. Other applications would be sought whose outputs match the outstanding required inputs. Cost per pound to orbit probably needs inputs of how much payload and how much cost. An analysis producing the pounds to orbit would seem to match one piece of this puzzle and, thus, a search for applications producing that result would be done. As illustrated in Figure 4.2, connections between applications generating discovered outputs and those consuming those outputs as their own inputs would be made and new inputs for the discovered applications would be added to the mix.

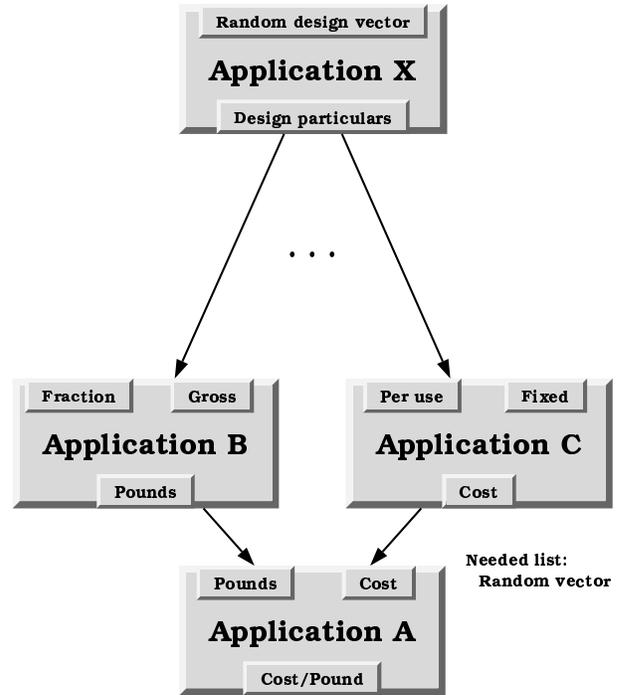


Figure 4.3: Reduction to Applications Requiring Only Random Inputs

The recursive cycle would continue in which applications producing outputs matching the currently needed inputs would be sought and, as those applications are found, their needed inputs would be, again, added to the mix. It is then proposed that, given the sufficiently rich environment that has been assumed, there will come a point, illustrated in Figure 4.3, at which the set of needed inputs will have been reduced to those that can be guessed, probably on a random basis, and, at that point, the process of solution construction can terminate.

Inputs that can be guessed will be those that specify an arbitrary, but semantically valid (though not necessarily good) design. For example as shown in Figure 4.4, a classic axi-symmetric propellant/oxidizer rocket engine is largely specified by a series of cross-sectional areas along an axis, in particular, the cross-sectional areas of the combustor, throat, and skirt. Some random number sets for these three key parameters will result in a good engine, most will not; but all such sets of three numbers result in an engine design whether or not it is good or bad. It may be that applications applying heuristic tools to randomly selected inputs may be formulated so that the range of a random design is not good to bad (with a preponderant emphasis on bad), but from good to adequate. This is illustrated in the figure with the constrained vector formulation: the combustion chamber and skirt areas are non-dimensionalized by the throat

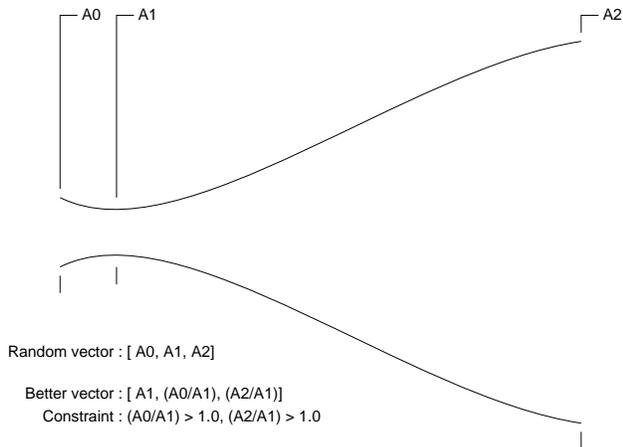


Figure 4.4: A Rocket Motor Design Application Requiring Random Inputs

area and constrained to be greater than 1.

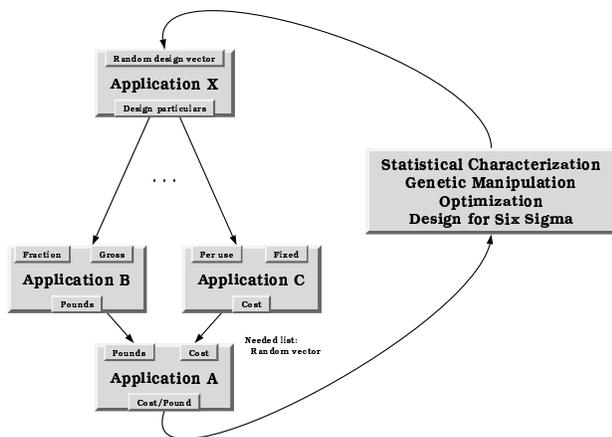


Figure 4.5: Application of Optimization Control

For the purposes of the SolveYourself method, the reduction of the problem to one needing randomly selected inputs only is a necessary and sufficient condition. As shown in Figure 4.5, an optimizing solution generator can be applied to the constructed application graph to pick a random starting point, analyze the result, and correct the design. A great deal of existing work in optimization methods, statistical characterization, genetic algorithms, and the like is applicable to this part of the SolveYourself method.

The algorithm of solution building has long been known and practiced in another form: that of program building that has been exercised by linker/loader programs for a half century or more. The linker/loader problem is more definite. An initial program module is inserted providing at least a key entry point, the **main** program, and entering a

list of needed entry points to complete the program. Other input modules and libraries are searched for pieces providing these entry points and, as those pieces are added, their internal tables add more entry points that they, in turn, need to operate. Many of those entry points will already have been identified; however, some may be new and cause yet another round of searching. Ultimately, linker/loader programs satisfy every such entry point request or report a failure to produce an executable program.

The search phase of the SolveYourself system is fundamentally the same as that of the linker/loader. The difference is that the SolveYourself system will, almost certainly, encounter ambiguities less easily resolved than those of the linker/loader operation. It may be that several applications will produce overlapping sets of currently-required inputs. Determining which application should be selected to provide the output meeting the needs of the required input may be challenging. On the surface it would seem that the confidence expressed by each competing application in its outputs would have some role in such a decision; however, an application's confidence in its output might depend upon its confidence in its inputs. A challenging exercise in disambiguation is almost certain to result. A SolveYourself solution must be prepared to maintain many possible, and perhaps complete, solution graphs so that rules and analyses may be applied across the constructed wholes to select among apparent equals.

#### 4.1 The Fly in the Ointment: The Presumption of Application Reliability

There is an implicit assumption underlying this entire proposal: it is that wrapped, served applications are, in fact, reliable. This is a fact not entirely in evidence in the case of many advanced analysis codes. For example, many current computational fluid dynamics codes do not simply read in a trivial input set and run reliably to a converged solution; instead, they rely on the expert care and feeding of the discipline expert that tweaks and pats and encourages the code toward an answer, and then casts a critical eye on that answer just to be certain.

There are a number of answers to this admitted weak point.

1. The PIA implementation already provides an event mechanism that wrappers may use to request attention from a discipline expert when usual events occur. The summoned expert can then intervene and provide the needed care and attention.
2. The PIA implementation already implements the ability to note when such failures occur and stop the flow

of solution for the affected problem configuration. If those failures occur in only some configurations, others are still able to soldier on. The net result will be that a portion of the design space will be blocked out as being untenable for reasons that are entirely auditable.

3. As will be expounded on shortly, the introduction of automated integration technology may allow disciplines to be narrowed and simplified in their focus. It is hoped that such simplifications might lead to more reliable, higher-fidelity applications.
4. The PIA application wrapper provides the place where needed expertise can be encapsulated. This is, indeed, one of the long term goals of the PIA effort: to capture the discipline expertise so that the ability to run applications will no longer retire with the people having those abilities.
5. Finally, unreliable applications are neither the fault nor the responsibility of the integration system. If some problems defy reliable analysis, then the expectation that finely-tuned, cost-effective, reliable, high performance systems can be achieved is unrealizable. We will need to order a great deal of chewing gum and bailing wire as our risk mitigation strategy.

## 5 The World Beyond the Solution

The discussion to this point considers only “analysis” applications that directly participate in the automatically-formed solution graph; however, the postulated “sufficiently rich” PIA collective can easily introduce many sources of relevant information that may not have a direct, participative role in the solution process. Specifically, many sources of experimental, empirical, or other data which document situations similar to portions of the problem at hand may exist and be automatically discoverable by the direct participants through PIA technology.

These sources of relevant data will likely be read-only in nature and will represent configurations fixed in time. Thus, these sources cannot be directly incorporated into one or another configuration of the solution process. As illustrated in Figure 5.1, this information can still contribute to the solution by providing starting or datum points to which the automated solution participants may refer.

Consider as an example of this a CFD analysis code that has been coupled into an automatically-generated solution graph. The inputs of the analysis may all have been syntactically satisfied by the solution graph generation process: gridded geometry might come from this predecessor

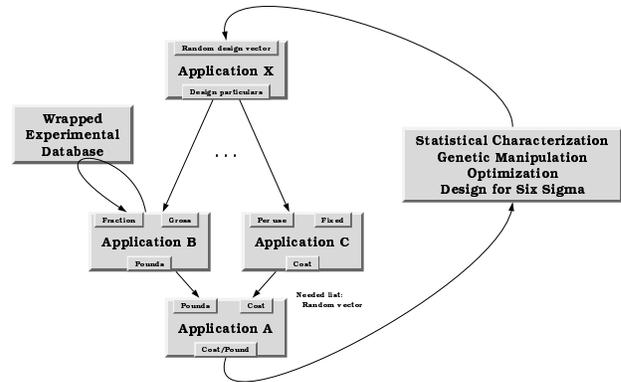


Figure 5.1: Use of Relevant Archives of Experimental and Other Data

or application, far-field boundary conditions from another predecessor application, and so on. Despite this technical satisfaction, the CFD code might still need some example starting point; it might need some initial solution taken from a similar problem which it can then iterate on computationally until it converges to a solution of the problem actually posed. It is in such an area that the relevant, non-participatory resources of the collective might still contribute: the CFD code (or, more accurately, the PIA-conformant wrapper of the code) could search the collective for flow field information of the desired kind and, finding such information, could discriminate its choices based on contextual issues such as the far-field conditions, geometric similarities, and the like.

The ability of solution participants to search the collective for relevant, but not directly participatory, information enhances the value of those static resources and warrants the introduction of those resources into the collective. The mountains of archived data – propulsion systems data, flow field data, airframe data, structural data, dynamics data, and the like *ad infinitum* – suddenly gains enormously in value as automated touchstones for current solution systems.

Finally, it might be imagined that the contributing information sources could extend beyond the range of static, pre-existing assets to encompass newly-generated sources requested by the solution process. Continuing the previous CFD example, suppose that a starting point is needed and that a search of the collective provides no sufficiently relevant example: the same PIA event mechanism that allows a wrapper to request assistance could also allow a sufficiently intelligent wrapper to request the performance of an experiment or other action to provide the needed information. The solution process could be suspended until the information is provided, or it could be terminated (perhaps just for that particular configuration) if the event response

indicates that no such information will be forthcoming. In this way the selection of experiments and other activities can become truly directed by the actual needs of the solution process, rather than simply being a guess at what data points may some day be needed.

## 6 Benefits of the SolveYourself Method

The key benefit of Autonomous Solution Systems (ASSs, the SolveYourself method) is that it applies the relentless, plodding stupidity (and ever-increasing speed) of the computer to a problem that is often mishandled by people.

### 6.1 Automation of the Teaming Process

The faults of people in the teaming arena are many. To start with, few people, if any, have the capacity to internalize the entirety of a system as complex as a launch vehicle or a high-performance air transport craft. Teams are inevitably formed to ameliorate this fact, usually in the face of political barriers disallowing the participation of some and ignorance unknowingly eliminating the participation of others. The fidelity of information transfer, even between the lowest, most technically competent levels, is often poor, while the fidelity of such processes at the highest team levels is customarily much worse.

ASSs eliminate people from this process and substitute machine-based teaming for human-based teaming. Within a PIA collective, there is no question of politics. Applications are not eliminated because they come from the wrong side of the router. If an application is served within a collective, it is trusted; its expression of confidence in the numbers it generates is taken as valid, without snicker or scowl. Furthermore, the search of a collective is exhaustive: an application does not go unnoticed because headquarters didn't know it had a group in Peoria that could handle this.

Because of the advances made by PIA-developed technology, the fidelity of information propagation within an automatically-generated solution is unerring: dimensional systems are never inadvertently mixed, design configurations never mismatched, digits are never reversed, bad data is never slipped in because the person who knew it was bad retired eleven years ago. Furthermore, the top decision maker of an automatically-generated solution is as fluent and accurate in the design information as those handling the lowest-level transfers.

### 6.2 Automatically-Generated Audit Trail

By using PIA technology already in place, an ASS approach would automatically generate a machine-auditable trail of both the examined designs and the process of assembling the method of solution. This trail would extend not only from the final, proposed answer to the first random guess, but would include all of the branches examined and rejected as being less than optimal.

### 6.3 Zero-Bias Solution Analysis

The application of ASS technology brings with it the potential, indeed the likelihood, that previously unconsidered formulations of any given problem will be considered. An ASS will not know that we always go to this division to solve this problem; while an ASS will see the capabilities of that division to bridge some particular solution gap, it will, in its generation of the potentialities of the solution graph, also find any other application paths that might bridge the same gap. Assuming some confidence-computation approach to graph disambiguation, it might well discover that the alternatives are more attractive, even though they might be more obtuse. Furthermore, as new capacities appear within a collective, the solutions that were identified yesterday might be different from the solutions that are mapped out tomorrow.

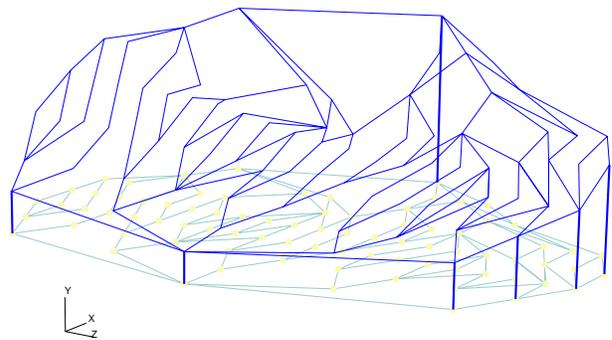


Figure 6.1: Confidence Levels Developed from an Assembled Application Graph

### 6.4 Automatic Identification of Weak Technology Areas; Risk Assessment

Another thing that might fall out as a byproduct of ASSs is the identification of technology areas that need work. The solution graph will map the path from random input to computed output. Confidence numbers will be associated throughout the graph as each component application node adds its expertise to the ever-improving analysis. Relatively simple analysis of the progression of confidence will reveal to people where further technology development is

needed and, thus, where to focus their further resources. This may be considered a form of risk assessment since it identifies those areas in which a given solution doesn't entirely know what it is doing.

An attempt is made to illustrate this concept in Figure 6.1 which plots solution confidence vertically over a hypothetical solution graph. Ideally, confidence should rise smoothly from initial to final node along each of the paths connecting those two points. Plateaus, gullies, and the like represent portions of the solution that are not performing at the same level as the overall solution.

### 6.5 Improved Utilization of Discipline Experts

ASSs eliminate the need for people to participate in teams by automating the team process. This makes more time available to discipline experts to advance the disciplines in which they are expert. The hypersonic inlets experts will not have to devote 2 hours on Monday and Thursday mornings to attending the team meetings, nor will they spend the full day every month going over the current design review, nor will they spend time on the phone with the combustor experts straightening out which flight condition that set of files was for. Instead, all of that time will be returned to the useful purpose of making their inlet analysis better.

### 6.6 Improved Maintenance of System-Involved Software

Use of the PIA collective technology also means that when the discipline experts have come out with the latest wrinkle in what their code can do, everything does not come to a halt while they repackage and redistribute their code to all the consuming points, those consumers update or re-install their copies, and everybody is retrained on what the new wrinkle does. Instead, the discipline server may not even have to pause while the link is changed to the revised application code. At most, an orderly pause may be needed to update the served wrapper. Solutions in progress may not realize that improved analysis is being received; however, a revision of the solution based upon the improved discipline capacity is an automatic process, requiring just another double click and, perhaps, a referral to the best that had been obtained in the previous solution.

### 6.7 Narrowing of Discipline Focus

Because of the transfer of integration tasks to an automated process, it may be possible to narrow and focus discipline research to more elemental problems. For example, instead of producing a single analysis of a complete inlet system,

it may be possible to focus available effort on a higher-fidelity analysis of a single inlet segment and leave it to the integration system to couple those elements into a complete inlet system analysis. This does, in this example, introduce the seemingly new discipline of flow physics transfer across a boundary; however, this discipline was implicit when one attempted the inlet system analysis in the first place. By relying upon integration technology, one complex discipline may be reduced to two or more simpler disciplines in which equivalent resource expenditure will have a greater return on investment.

## 7 Rationale

Having considered all of these possibilities and supposed benefits, one significant question remains: why bother with ASSs? After all, commercial integration systems exist in which the technologist manually connects the dots from application to application. These systems have intuitive GUI interfaces for the integration task, have proven easy to use, and have produced real-world savings and benefits. Why bother automating this already successful technology?

The answer to this question is that ASSs will be necessary to assemble analyses of truly significant systems; reusable launch systems, advanced air transport vehicles, complete space exploration missions, and the like. The manual integration systems are only practical when applied to relatively simple systems which result in dot-connection processes of manageable proportions; perhaps ten or twenty applications cooperating to provide a system analysis of, say, a multi-stage turbine. When the number of applications to be integrated begins to increase from tens to hundreds, thousands, ten-thousands, and beyond to provide the analysis of a complete reusable launch vehicle system, the magnitude of the integration process will simply grow beyond the capacity of manual techniques. The probability that some member of the integration team will manually connect the wrong dot will grow to the point of near certainty. (Consider the integration of 1000 applications, each transferring 20 items on to other members of the integrated whole; if it is 99.99 per cent certain that each connection of an item to a receiving application is correct, then there is an 86 per cent probability that one connection somewhere in the integrated whole is wrong.) It is the automation of the integration process, with its inerrant dot-connection process, that will enable such system analyses to be conducted.

In addition to enabling the comprehensive analysis of complex systems, ASSs will further enable the re-analysis of those systems as needed new analytical capabilities are identified and introduced. Even if a complete complex system were analyzed through a massive, manual integra-

tion team effort, the idea that it would be economically re-integrated again and again and again as new component analyses became available is doubtful. Further, the idea that such re-integrations would be adequately documented so that it was clear just what analysis had been conducted to provide the final, acceptable answer is even more remote. ASSs enable documented, auditable re-integration of complex system analyses by applying the mindless, plodding stupidity of the computer to that task.

## 8 Summary

The existence of an ability for problems to meaningfully solve themselves has been postulated and a sketch of the methodology by which this might be done has been presented. The concepts of self-revelation and semantic infusion through class derivation developed through the PIA effort are key to this capacity, allowing the problem of solution generation/organization to be pursued in a manner like that of executable programming linking. Key benefits of Autonomous Solution System (ASS) technology include the following.

1. Automation of the teaming process; human foibles, inaccuracies, fantasies, and limitations are eliminated from the teaming process and the relentless, plodding, unerring stupidity of the computer is substituted.
2. Automatic generation of a solution audit trail; every decision point from initial method formulation to final, proposed design solution would be documented in a machine-auditable manner; all alternatives examined and rejected would be retained for re-examination of the decision process.
3. Elimination of bias from the solution approach; alternative strategies to each solution phase are dispassionately considered each time a new problem is posed.
4. Automatic identification of weak technology areas; analysis of confidence levels through a generated solution graph provides a basis on which to assess where capabilities are weak; this may form the basis of automated risk assessment.
5. Automation of team processes improves discipline-focused time utilization; discipline experts are relieved of teaming duties and therefore have more time to devote to discipline-advancing pursuits.
6. Reduced software maintenance; discipline software is served by a single server (or cluster of servers) associated with the center of discipline expertise rather than

distributed to all the consumers of that discipline software; revisions and updates can achieve near transparency.

7. Integration technology can enable narrowed discipline focus; it may be possible to abandon complex system disciplines in favor of more narrowly focused elemental disciplines; even though new integration disciplines are introduced, the higher levels of discipline clarity may result in an overall improvement in the return on investment.

Finally, the application of automation to the solution formulation task allows the extension of integrated analysis into realms beyond the reach of existing manual techniques.

## 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. Presented at the International Gas Turbine and Aero-engine 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] 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.
- [5] Theresa Louise Benyo. *Project Integration Architecture (PIA) and Computational Analysis Programming Interface (CAPRI) for Accessing Geometry Data from CAD Files*. Technical memorandum NASA/TM-2002-211358, National Aeronautics and Space Administration, Lewis Research Center, 21000 Brookpark Road, Cleveland, OH 44135, March 2002.

- [6] William Henry Jones. Project Integration Architecture: Formulation of Dimensionality in Semantic Parameters. Draft paper available on central PIA web site, March 2000.
- [7] William Henry Jones. Project Integration Architecture: Distributed Lock Management, Deadlock Detection, and Set Iteration. Draft paper available on central PIA web site, April 1999.
- [8] William Henry Jones. Project Integration Architecture: Inter-Application Propagation of Information. Draft paper available on central PIA web site, December 1999.
- [9] William Henry Jones. Project Integration Architecture: Formulation of Semantic Parameters. Draft paper available on central PIA web site, January 2000.
- [10] William Henry Jones. Project Integration Architecture: Wrapping of the Large Perturbation Inlet (LAPIN) Analysis Code. Draft paper available on central PIA web site, March 2001.
- [11] William Henry Jones. Project Integration Architecture: Implementation of the CORBA-Served Application Infrastructure. Draft paper available on central PIA web site, May 2002.

Nearly all material relevant to the PIA effort, including complete, class-by-class, member-by-member documentation, is available on a central server provided by 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 generation of the researchers involved and do not in any way represent a commitment of the Government of the United States, yada, yada, yada.