An Object Oriented Extensible Architecture for Affordable Aerospace Propulsion Systems

Gregory J. Follen

Computing & Interdisciplinary Systems Office

NASA Glenn Research Center
21000 Brookpark Rd.
Cleveland, Ohio 44135
gfollen@grc.nasa.gov
USA

Summary
Driven by a need to explore and develop propulsion systems that exceeded current computing capabilities, NASA Glenn embarked on a novel strategy leading to the development of an architecture that enables propulsion simulations never thought possible before. Full engine 3 Dimensional Computational Fluid Dynamic propulsion system simulations were deemed impossible due to the impracticality of the hardware and software computing systems required. However, with a software paradigm shift and an embracing of parallel and distributed processing, an architecture was designed to meet the needs of future propulsion system modeling. The author suggests that the architecture designed at the NASA Glenn Research Center for propulsion system modeling has potential for impacting the direction of development of affordable weapons systems currently under consideration by the Applied Vehicle Technology Panel (AVT).

This paper discusses the salient features of the NPSS Architecture including its interface layer, object layer, implementation for accessing legacy codes, numerical zooming infrastructure and its computing layer. The computing layer focuses on the use and deployment of these propulsion simulations on parallel and distributed computing platforms which has been the focus of NASA Ames. Additional features of the object oriented architecture that support Multi-Disciplinary (MD) Coupling, computer aided design (CAD) access and MD coupling objects will be discussed. Included will be a discussion of the successes, challenges and benefits of implementing this architecture.

Numerical Propulsion System Simulation

Today, propulsion engineers use what are called preliminary and conceptual design codes to numerically create and analyze commercial, military and rocket propulsion systems. Most of these computer codes were written in the 60's and 70's and many, if not all, are written in FORTRAN. For some time now, analyzing and building propulsion systems has been prohibitively expensive due largely to the iterative nature of designing, analyzing and testing of hardware before a final configuration is achieved. In order to reduce cost, risk, time to market, expand capability, assure accuracy to mission requirements and increase confidence in designs, innovative ways have to be found to numerically create propulsion systems that bring the design closer to the final configuration before hardware is ever built and tested.

The NASA Glenn led Numerical Propulsion System Simulation (NPSS) is a project funded by and partnered with NASA Ames targeted at impacting this need. NPSS set out to advance the state of the art in propulsion modeling as well as create a common architecture to numerically model propulsion systems. NASA Ames embarked on developing the parallel and distributed computing platforms needed by such simulations.

The current state of the art in propulsion modeling centers on the use of 0 Dimensional preliminary and conceptual design methodology. However, NPSS wanted to look beyond the current ways propulsion systems were designed and created. NPSS dreamed of a system that allowed an engineer the flexibility to numerically assemble an engine using 3-Dimensional components or any combination of 0,1,2,3 Dimensional component codes. The "plug-n-play" or "substitute at will" concept captures the essence of this goal. Figure 1 embodies this concept.

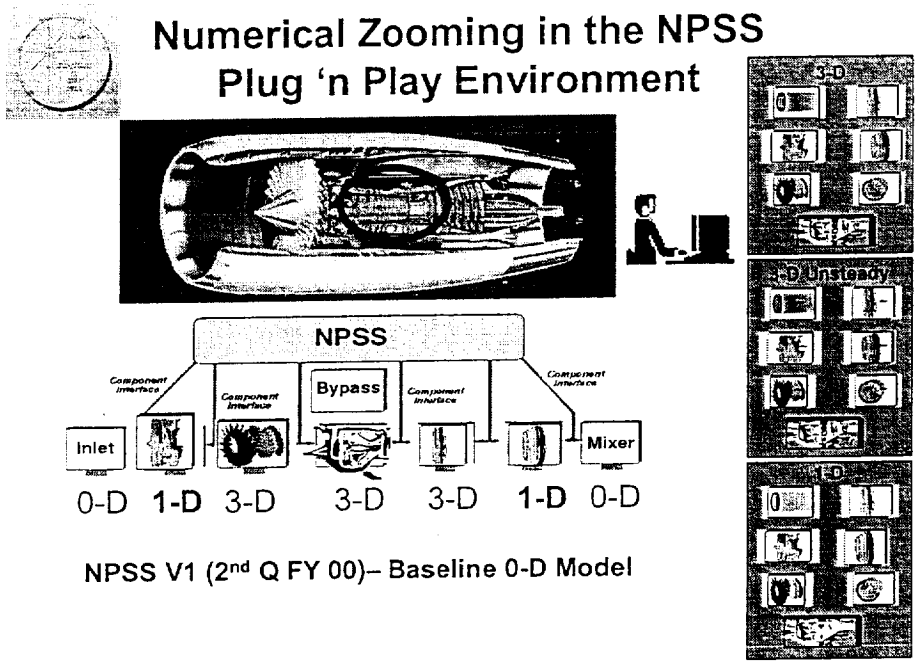

Figure 1.

With this in mind, an object-oriented architecture was designed and laid out to fulfill this vision. The NPSS object-oriented architecture allows an engineer to numerically assemble a propulsion system comprised of differing dimensionality component codes (Numerical Zooming), different disciplines (MD coupling), all irrespective of the computing platforms these codes execute on while producing results on cost effective computing platforms overnight. The first deliverable within the NPSS Architecture is NPSS V1.0. Although V1.0 preserves the traditional preliminary and conceptual design methodology for designing engines that is the state of the art today, it was created to move the state of the art in propulsion system modeling into the future. As such, NPSS V1.0 is an object oriented preliminary and conceptual design code used by aerospace engineers to predict and analyze the aero-thermodynamic behavior of commercial jet aircraft, military, and rocket engines. However, it is more than this, as it has designed into it the infrastructure supporting Numerical Zooming to higher dimension codes and coupling to

differing discipline analysis. As the state of the art in propulsion modeling moves into the future away from a strict adherence to 0 Dimensional analysis towards a mixture of 0,1,2,3 Dimensional codes, the same NPSS' architecture exists to support this maturity in modeling.

The NPSS architecture is pictorially represented by figure 2. The architecture is open and extensible. To this end, the architecture exploits the capabilities of object-oriented programming (inheritance, polymorphism, and encapsulation) as well as modern object-oriented concepts including frameworks, component objects, and distributed object standards.

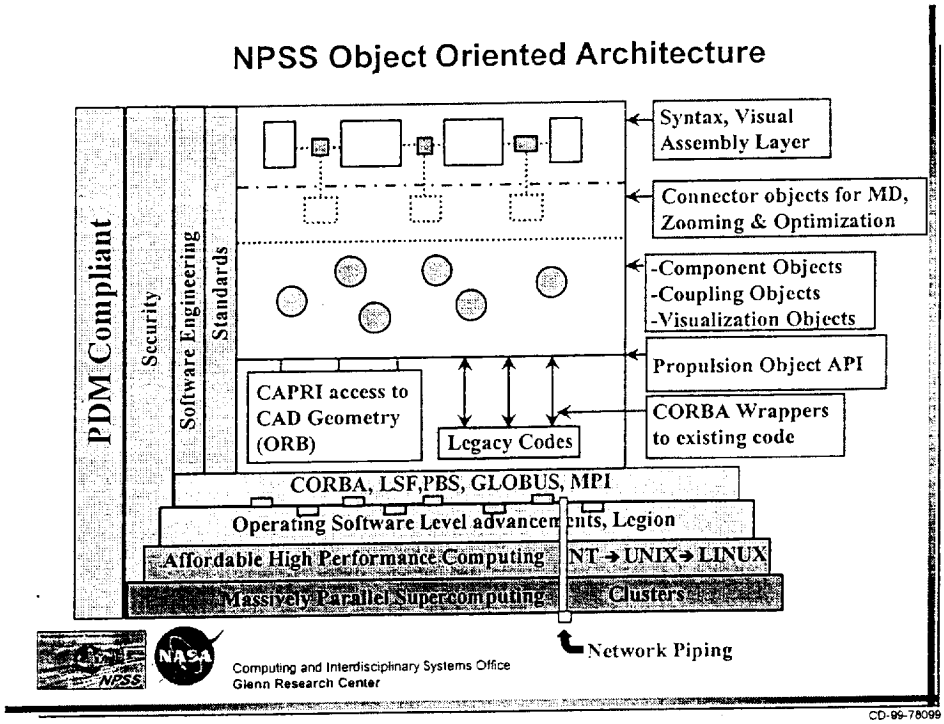

Figure 2.

Design Philosophy

The NPSS Architecture was designed following a hybrid object oriented design philosophy. The early work by I. Jacobson and G. Booch were followed where appropriate and extended by experiences known within the NASA culture. An overall philosophy for NPSS was to view the architecture from a leap-frogging approach, purchase from the commercial sector what you could, minimize commercial licensing and build from scratch that what you must. Whatever benefited the project from new and innovative hardware or software was to be incorporated into the architecture as quickly as possible without a huge development effort and without disturbing the quality and stability of the current system. To make the advances in propulsion design, time could not be wasted on re-checking answers, re-writing code and re-designing entire sections of the architecture. This was the fundamental reason the object-oriented paradigm was chosen. Beliefs then, and proven now, demonstrate that the object oriented design methodology was correct.

NPSS Architecture

Referring to figure 2. above, there are fundamentally three main areas of the architecture. These are: the Interface Layer, Object Layer and the Computing Layer. Within the Interface Layer, a command and a visual interface exist. The Object Layer contains the fundamental engineering specifics for propulsion systems and the appropriate support objects needed by propulsion systems such as access to geometry and legacy FORTRAN codes used by many, if not all, propulsion companies. Last but not least, the Computing Layer exists on which to deploy propulsion system simulations. This last layer, Computing, is and has been the most dynamic over the last ten years and continues to change about every 18-24 months.

Interface Layer

From the beginning of the architectures' development, the priorities were to get the engineering and physics right and then add a visual interface later. Given this, the main interface to NPSS has been a command line. However, do not assume that this is a simplistic interface to NPSS. On the contrary, the command line and its suite of syntax are quite elegant, mature and sophisticated. Two versions of the command interface exist:

Batch: npss [*-options*]file1 *file2* . . .
Interactive: npss [-i][*-trace*][*-options*]file1 *file2* . . .

Contained within the file1 is the actual NPSS syntax that defines the propulsion system to be designed and analyzed. The language used here is C++ like but not pure C++. Early exposure to pure C++ as the syntax changed the direction to create a C++ like syntax. This change allowed an easier and early adoption of NPSS. While most engineers wanted a FORTRAN language, many of the concepts envisioned fell victim to FORTRAN's language syntax. The syntax itself has many features of a programming language and indeed, a feature we've added is an NPSS syntax to C++ converter. This feature allows code first developed with the syntax to be later compiled as part of an executable library available for later use. A sample of the syntax looks like the following.

```
Model BWB {
        Element FlightConditions AMB0 {. . .}
        Element Inlet Inlet {. . .}
        Element Fan Fan{. . .}
        Element Compressor Compressor{..}
        Element Combustor Combustor{. . .}
        Element Turbine Turbine {. . .}
        Element Nozzle Nozzle(. . .}
        linkPorts ("FlightConditions.Outlet", "INLET.F1_I", "FL0",. . . . . );
}
```

The syntax has programming constructs such as: The ability to declare new variables that are combinations of other variables, comments, If-then-else, do while's, arithmetic functions: *, /, +, -, exponentiation, logicals, >, <, =,...., etc.

Recently, a visual front end to NPSS has been under development. The visual front end communicates with the NPSS system through the command interface as just described. An early view of this interface is included as shown in figure 3. Visually speaking, NPSS provides the ability to assemble and connect a propulsion system together and then execute this simulation as well as store or archive it as necessary. It is the author's belief that in order to maintain

flexibility, maturity of NPSS and advancement to its capabilities, a visual interface and a command interface must always exist separately. Once the visual interface becomes part of a code, a violation to the integrity of the original intent of the code has occurred and can never be recovered. Even as future interfaces emerge such as voice activation, screen sensing and even optical movement or heads up displays, in order to break off the current visual interface and make use of the futuristic interfaces, a command interface will need to exist.

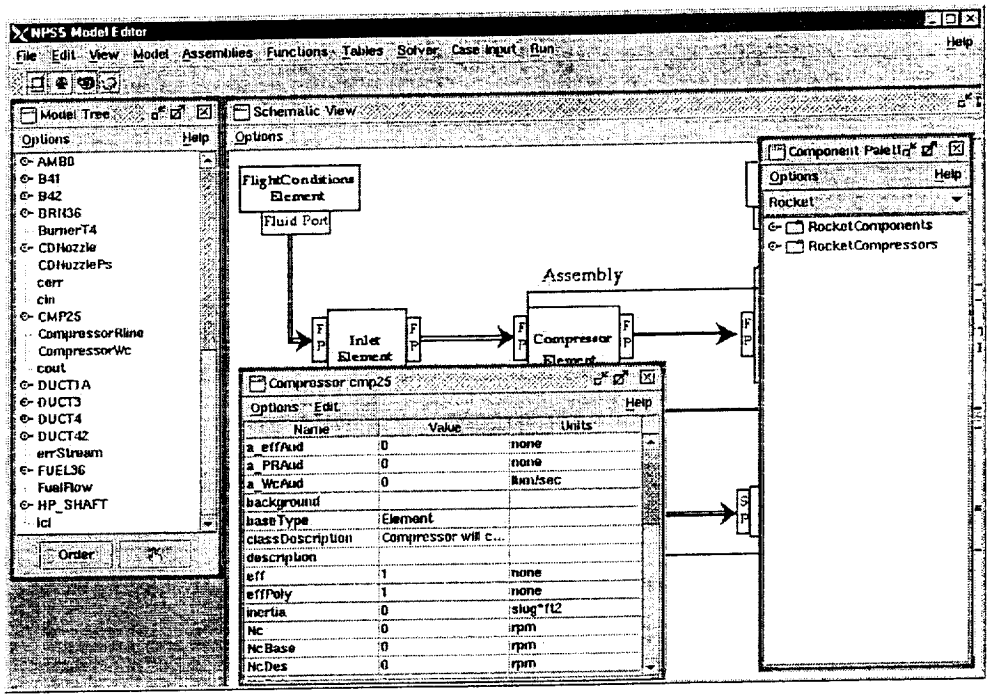

Figure 3.

Object Layer

The development of the object layer has garnered the most attention and development. This is where all the propulsion objects exist both for airbreathing and rocket engines. Additionally, this is where all the infrastructure objects are for moving information from one element to another, for accessing object codes (FORTRAN, C, C++,...) through CORBA on other machines and other address spaces. The numerical zooming, code coupling and security infrastructure also exists here as well.

Propulsion and Rocket Objects

NASA Glenn has populated this layer with airbreathing, rocket and to a lesser extent ground based power objects. What is marquee about this architecture is that the infrastructure contained within the NPSS syntax is the same for airbreathing, rocket and ground based power objects. While the NASA Glenn led team came together to define what a common set of airbreathing, rocket and ground based power objects are and defined their numeric behavior, this does not mean that the objects' behavior and characteristics cannot be changed on demand. On the contrary, the ability to change or extend the objects behavior is central to the use of the object-oriented paradigm. The objects provided can be used as they are or can be changed based upon appropriate need. Additionally, the developer is assured that the object has been tested and proven to be accurate. So, any abnormal behavior is due solely to the new features just

introduced by the developer. The basic objects used within the NPSS Architecture for 0 Dimensional and 1 Dimensional analysis are:

- Elements
  - –Primary building blocks connected together via Ports
  - –Perform high-level calculations
- Subelements
  - –Interchangeable secondary building blocks that plug into Elements or other Subelements
  - –Perform detailed calculations
- Flow Stations
  - –Responsible for thermodynamic and continuity calculations
  - –Access the thermodynamic packages
- Ports
  - –Used to connect Elements together
  - –Five types (Mechanical, Fluid, Fuel, Thermal, Data)
  - –Directional in nature (i.e., outputs connect to inputs)
- Tables
  - –Organized set of numbers that relate n-dimensional inputs to one or more outputs
  - –Support linear and second or third order LaGrange interpolation
  - –Support fixed value end-points or extrapolation (linear/2nd/3rd order LaGrange)
  - –May be used at any location a function is called and vice-versa

Of particular note, in this object definition, is that there isn't a reference to anything related to propulsion. The NPSS Architecture's object structure, as defined, has allowed its general usage amongst airbreathing, rocket, fuel cell and ground based power propulsion by the writing of the appropriate functional objects. The author believes there are more applications to come.

Zooming, Code Coupling

In order to recover the wealth of investment in current FORTRAN, C, or other codes, NPSS has adopted and developed a Common Object Request Broker Architecture (CORBA) interface to make it appear as though these codes are actually C++ objects within the architecture. While NPSS cannot gain complete control over these codes, it does provide three common procedures for integration. These procedures are currently GET a variable, SET a variable, and EXECUTE and are available no matter what the particular focus of the code you are accessing or in NPSS terms "zooming" to. The zooming infrastructure has been successfully demonstrated between an NPSS turbofan model and a 1 Dimensional high-pressure compressor code as pictorially represented by figure 4. A similar zooming accomplishment has also been completed between an NPSS Expander Cycle Engine model and a 1 Dimensional Pump code called PUMPA.

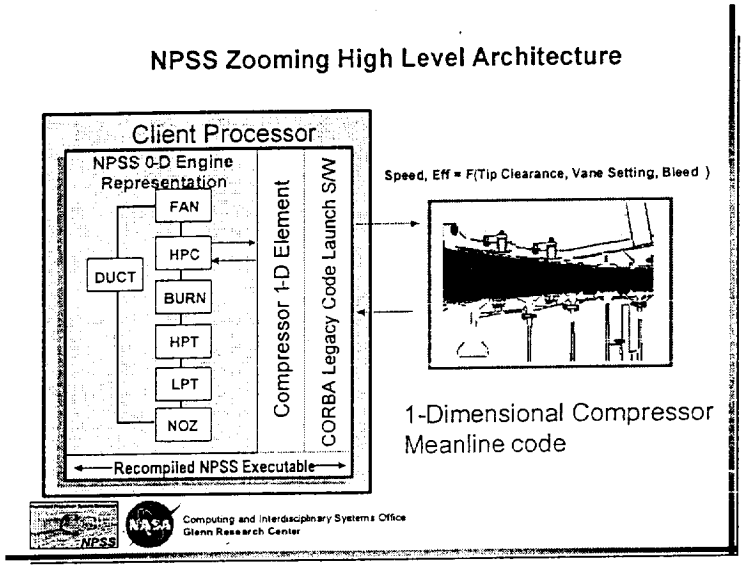

Figure 4.

Distributed, Parallel Computing

The basic internal communication scheme used by the NPSS Architecture for moving data across address spaces and separate machines is through its CORBA interface. This is a point-to-point concept of distributed computing and coupling of codes. Leveraging CORBA and its associated Security (CORBASec) software has proven quite useful. From a parallel processing sense, the NPSS Architecture has adopted the NASA Ames' Grid Computing software called GLOBUS. The GLOBUS software can be thought of as a scheduler of schedulers since it is 'aware' of scheduling software such as Load Sharing Facility (LSF,Platform Computing), Portable Batch System (PBS,NASA Ames), Load Leveler (IBM). The NPSS Architecture makes an assumption that any 3 Dimensional Computational Fluid Dynamics (CFD) code, structures codes that will be integrated into an NPSS Simulation, already uses one of the above schedulers as well as a particular message-passing library such as Parallel Virtual Machine (PVM), Message Passing Interface (MPI) or some derivative of these. For NPSS' needs, the Grid Computing software GLOBUS, needed to be aware of CORBA based simulations in the same way that it is aware of LSF, or PVM based simulations. The NPSS team has developed a CORBA interface to the GLOBUS services to support the NPSS Architecture. NPSS' goal has been to deploy complex propulsion simulations that can be solved in an overnight timeframe in less than fifteen hours on cost effective computing platforms. The corresponding Architecture goal is to deploy these subject simulations on any computing cluster with minimal to no changes to the codes.

Coupling of Codes & Geometry Application Program Interface (API)

In developing the appropriate objects to support code coupling both from a single discipline and multi-discipline perspective, a prototyping activity was set forth through a contract. The contract specified a to prototype the needed objects to couple a 3 Dimensional structures with a 3

Dimensional fluid code together and thereby provide guidance to a design that would encompass most of the objects needed to support coupling of codes. Once the NPSS project understood what made sense to do, the team could then begin the formal development of the C++ objects and make this part of the NPSS Architecture. While this work is continuing, current objects include: single precision variable object, structured grid object, structured interface object, meta variable object, FORTRAN character object, FORTRAN callable API, and a CORBA Object called ForeignElement.

The NPSS Architecture needed a common geometry API that interfaced to the current suite of commercial CAD vendors. It was already known that differing 3 Dimensional codes acquired their respective geometry in different ways which introduced the potential for errors in the analysis as the geometry was not interpreted the same way by all authors of codes. To address this issue, an activity was started with Massachusettes Institute of Technology (MIT) to create a common interface for reading geometry from Unigraphics, ProE, CATIA as these represented the most common CAD systems within the propulsion community NPSS worked with. The result of this work is a library/API called CAPRI. If the code developer adopts the CAPRI API then geometry from UG, ProE, CATIA and other CAD systems can be read directly. Work is ongoing to implement the Write function. No status is provided here on the Write function.

Computing Layer

The High Performance Computing and Communication Program that funded NPSS had as one of its goals the development of massively parallel computers with high speed networks. Knowing this and realizing that early computing platforms of this type would be volitale, the NPSS Arhitecture was designed to 'leap-frog' this computing technology with no or minimal penalty to the NPSS applications. Where it was necessary for an application to port to an architecture, NPSS intended to use CORBA to reach that application. The developers of the NPSS Architecture wanted minimal proprietary computing presence so as to be able to jump from one parallel or distributed computing platform as needed. The thought of porting codes to particular architectures and then port them again all the while re-validating these codes was to be avoided.

NASA Glenn's participation in the development of parallel computers and networks focused on cost effective clusters. Originally, a thirty-two node cluster of IBM 560's with multiple networks was assembled from commercially available UNIX machines. These systems were ugraded to IBM 590's but never grew beyond the original 32 nodes. Following this cluster came a 128 node Pentium PC cluster comprised of 64 dual processor Pentium 400 Mhz systems running LINUX. Both these systems were batch oriented with the resources controlled by Platform Computing's Load Sharing Facility (LSF). A partnership arose between NASA Glenn, Platform Computing and a commerical propulsion company to support parallel applications within LSF and to develop a multi-cluster capability. Both these features exist within LSF today. Currently, the NPSS project is moving to adopt the GLOBUS software that can co-exist with LSF and has support for scheduling applications built on differing batch schedulers such as the Portable Batch Scheduler (PBS).

Software Engineering Principles

NPSS made an assumption that in order to see its Architecture and code used within the US Industry, adherence to an identifiable and recognizable software development process was

mandatory. Even within a research center, software engineering principles have a place. In many research efforts, software engineering practices are a 'post development' activity. Many developers always seem to have time to go back and fix what they didn't have time to design for or around from the beginning. This was not the case with NPSS. From the start, software development plans, configuration management plans, and verification and validation plans and design plans were developed and used. At a minimum, any software development effort should have configuration management and some form of verification and validation at the subsystem and full system level. Without these phases, research codes never become anything but single user research codes. Any development effort sponsored by the AVT should include a presence of software engineering practices as appropriate for this community.

Along with a sophisticated yet manageable software development process, NPSS adopted an incremental release process by which bug fixes and urgently needed enhancements found their way into smaller or incremental releases rather than waiting for full releases of the software. Regression testing and documentation are still maintained within this incremental release process and in fact are more manageable.

Benefits to Date of NPSS

Early indicators on the benefits of using NPSS reveal a 55% reduction in the time to perform engine system simulation throughout the product life cycle. Additionally, expectations include a 50% improvement in business processes with partners and customers.

From zooming, a reduction factor of 10 was achieved by using the NPSS architecture to integrate high-fidelity compressor code into a system model. What normally took two days was achieved in two hours making a simulation do able whereas before it was possible but not practical.

Summary & Appropriateness to AVT

The words used in the theme of this AVT meeting are very similar to the words used in the goals and approach in the development of NPSS. The AVT theme states, "The defense of NATO requires a new paradigm in the development and deployment...", " essential to achieving the cost and time reductions needed to field new and improved...". NPSS' goal is to reduce time and cost in development of new propulsion systems while increasing confidence and reducing risk in achieving a final design. The NPSS Architecture emerged to impact airbreathing propulsion in the ways mentioned above. However, soon after its first incremental release, NPSS' potential use to impact space propulsion and ground based power also was realized. The NPSS architecture was re-used to model rockets, ground based power systems and even fuel cells by populating only its object layer with the behavior needed for space propulsion, ground based power and fuel cells. The remaining architecture was reused. The process by which NPSS was built is noteworthy. Designing a system that combines a production phase with prototyping and deployed on an incremental release schedule, provides for early access to fixes and new features that ultimately lead to the stated goals of reducing risk and reducing the time to final design. It is the opinion of the author that similar gains to AVT are available in building Affordable Weapons systems for NATO as are now being realized within NPSS' propulsion and power community.

Acknowledgement
The author would like to express appreciation to Cynthia Naiman, Isaac Lopez, Robert Griffin, Desheng Zhang, Scott Townsend, Chris Beins and to the members of the entire NPSS team, composed of representatives from the NASA Glenn Research Center, US Engine Manufacturers, US Aircraft companies, DOD and university representatives, for their invaluable contributions to the NPSS project. The author would also like to express appreciation to the NASA Ames Management, specifically Catherine Schulbach, William VanDalsem and Jerry Yan, of the High Performance Computing and Communications Program for sponsoring this work.

References

1. Lytle, J. K., Follen, G. J., Naiman, C. G., Evans, A. L., Owen, A. K., Veres, J. P., and Lopez, I., Numerical Propulsion System Simulation (NPSS), 2000 Industry Review, NASA Glenn Research Center, October 2000.
2. Townsend, S.E., Using External Codes With NPSS, NASA Glenn Research Center, October 2000.
3. Sang, J., Follen, G. J., Kim, C., and Lopez, I., Townsend, S. E., Enhancing the Remote Variable Operations in NPSS/CCDK, NASA Glenn Research Center, October 2000.
4. Lytle, J. K., Follen, G. J., Naiman, C. G., Evans, A. L., Veres, J. P., and ET. AL., 1999 Numerical Propulsion System Simulation Industry Review, NASA/TR 209795, NASA Glenn Research Center, October 6, 1999.
5. Lytle, J. K., Numerical Propulsion System Simulation: An Overview, CAS 2000 Workshop/The Ames Research Center, February 15- 17, 2000.
6. Claus, R., Follen, G. J., Haimes, R., and Jones, W., CAPRI - Computational Analysis Programming Interface/ A CAD infra-structure for Aerospace Analysis and Design Simulations, NASA HPCC/CAS Workshop/NASA Ames Research Center, February 15, 2000 - February 17, 2000.
7. Follen, G. J., and auBuchon, M., Numerical Zooming Between the NPSS Version 1 and a 1-Dimensional Meanline Design Analysis Code, AIAA 99-SL-217/USA-35, September 5, 1999 - September 10, 1999; 14th International Symposium on Airbreathing Engines (XIV ISABE), Florence, Italy.
8. Hall, E. J., Modular Multi-Fidelity Simulation Methodology for Multiple Spool Turbofan Engines, NASA HPCC/CAS Workshop/NASA Ames Research Center, February 15, 2000 - February 17, 2000.
9. Sang, J., Kim, C., and Lopez, I., Developing CORBA-Based Distributed Scientific Applications From Legacy Fortran Programs, NASA/TM 209950, CAS Workshop/NASA Ames Research Center,, February 15, 2000 - August 1, 2000.
10. Lopez, I., Follen, G. J., Gutierrez, R., Foster, I., Ginsburg, B., and ET. AL., NPSS on NASA's IPG: Using CORBA and GLOBUS to Coordinate Multidisciplinary Aeroscience Applications, NASA/TM 209956, CAS Workshop/NASA Ames Research Center, February 15, 2000 – February 17, 2000.
11. Townsend, S.E., Coupling ADPAC to NPSS for an EEE Simulation, NASA Glenn Research Center, October 1999.
12. Reed, J. A., and Afjeh, A. A., Computational Simulation of Gas Turbines: Part I, II - Extensible Domain Framework, ASME, International Gas Turbine & Aeroengine Congress & Exhibition, Indianapolis, Indiana, June 7, 1999 - June 10, 1999.
13. Evans, A. L., Naiman, C. G., Lopez, I., and Follen, G. J., Numerical Propulsion System Simulation's National Cycle Program, 34th AIAA/ASME/SAE/ASEE; Joint Propulsion Conference & Exhibit, Cleveland, OH, July 13, 1998 - July 15, 1998.
14. Hall, E. J., Delaney, R. A., Lynn, S. R., and Veres, J. P., Energy Efficient Engine Low Pressure Subsystem Aerodynamic Analysis, NASA CR-206597, 34th Annual AIAA/ASME/SAE/ASEE Joint Propulsion Conference & Exhibit, July 13, 1998 - July 15, 1998, Cleveland, Ohio, AIAA 98-3119.
15. Haimes, R., CAPRI - Computational Analysis Programming Interface, 6th Int'l Conference on

Numerical Grid Generation, University of Greenwich; Greenwich, UK, July 6, 1998 - July 9, 1998.
16. Lytle, J. K., The Numerical Propulsion System Simulation: A Multidisciplinary Design System for Aerospace Vehicles, NASA/TM-1999-209194, September 5, 1999 - September 10, 1999, 14th International Symposium on Air Breathing Engines sponsored by the International Society for Air Breathing Engines, Florence, Italy.
17. Foster,I., Tuecke, S., Kesselman, C., et al, www.globus.org