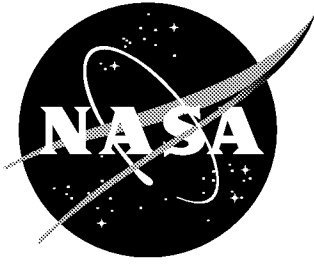


NASA/CR-1998-208731



Evaluation of Frameworks for HSCT Design Optimization

Ramki Krishnan
Computer Sciences Corporation, Hampton, Virginia

October 1998

The NASA STI Program Office ... in Profile

Since its founding, NASA has been dedicated to the advancement of aeronautics and space science. The NASA Scientific and Technical Information (STI) Program Office plays a key part in helping NASA maintain this important role.

The NASA STI Program Office is operated by Langley Research Center, the lead center for NASA's scientific and technical information. The NASA STI Program Office provides access to the NASA STI Database, the largest collection of aeronautical and space science STI in the world. The Program Office is also NASA's institutional mechanism for disseminating the results of its research and development activities. These results are published by NASA in the NASA STI Report Series, which includes the following report types:

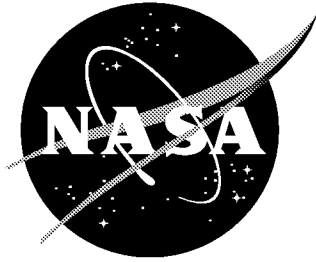
- TECHNICAL PUBLICATION. Reports of completed research or a major significant phase of research that present the results of NASA programs and include extensive data or theoretical analysis. Includes compilations of significant scientific and technical data and information deemed to be of continuing reference value. NASA counterpart of peer-reviewed formal professional papers, but having less stringent limitations on manuscript length and extent of graphic presentations.
- TECHNICAL MEMORANDUM. Scientific and technical findings that are preliminary or of specialized interest, e.g., quick release reports, working papers, and bibliographies that contain minimal annotation. Does not contain extensive analysis.
- CONTRACTOR REPORT. Scientific and technical findings by NASA-sponsored contractors and grantees.
- CONFERENCE PUBLICATION. Collected papers from scientific and technical conferences, symposia, seminars, or other meetings sponsored or co-sponsored by NASA.
- SPECIAL PUBLICATION. Scientific, technical, or historical information from NASA programs, projects, and missions, often concerned with subjects having substantial public interest.
- TECHNICAL TRANSLATION. English-language translations of foreign scientific and technical material pertinent to NASA's mission.

Specialized services that complement the STI Program Office's diverse offerings include creating custom thesauri, building customized databases, organizing and publishing research results ... even providing videos.

For more information about the NASA STI Program Office, see the following:

- Access the NASA STI Program Home Page at <http://www.sti.nasa.gov>
- E-mail your question via the Internet to help@sti.nasa.gov
- Fax your question to the NASA STI Help Desk at (301) 621-0134
- Phone the NASA STI Help Desk at (301) 621-0390
- Write to:
NASA STI Help Desk
NASA Center for AeroSpace Information
7121 Standard Drive
Hanover, MD 21076-1320

NASA/CR-1998-208731



Evaluation of Frameworks for HSCT Design Optimization

Ramki Krishnan
Computer Sciences Corporation, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-20431

October 1998

Available from:

NASA Center for AeroSpace Information (CASI)
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 605-6000

Evaluation of Frameworks for HSCT Design and Optimization

Ramki Krishnan
Computer Sciences Corporation

CAPSS Contract NAS1-20431

This report is an evaluation of engineering frameworks that could be used to augment, supplement, or replace the existing FIDO 3.5 (Framework for Interdisciplinary Design and Optimization Version 3.5) framework. The report begins with the motivation for this effort, followed by a description of an “ideal” multidisciplinary design and optimization (MDO) framework. The discussion then turns to how each candidate framework stacks up against this ideal. This report ends with CSC’s recommendations as to the “best” frameworks that should be down-selected for detailed review in the coming months.

It needs to be emphasized that this report evaluates select frameworks based on a number of factors. Some of these factors were adopted from a manifesto developed by Jim Rogers (with help from Andrea Salas) at NASA Langley. Other aspects were issues that the CSC developers and engineers felt strongly about, and thought needed to be among the primary criteria. It is hoped that the issues considered by this report are reflective of the concerns expressed by other groups at NASA Langley

who are desirous of utilizing an framework to facilitate their work.

The topic of the “ideal” framework sparked many discussions between NASA researchers and CSC, and among CSC engineers. As such, this document is a compendium of issues and concerns expressed during those deliberations.

Long, long, ago.....

The current framework (i.e. FIDO 3.5) had its origins 4 years ago and was primarily intended as a prototype framework to study the implementation of a MDO methodology. In that sense, FIDO has accomplished its modest goals. Many discipline codes have been successfully integrated into the framework, the framework has been utilized for complicated interdisciplinary analyses, and a simplified optimization procedure was validated recently using the framework.

To satisfy today’s design demands, FIDO’s performance requirements have evolved from demonstrating the viability of an optimization methodology for a simplified HSCT-type configuration to being able to handle this question: How can the framework help MDO practitioners utilize their codes and procedures to evaluate optimization schemes? This is an important goal for MDO researchers; complex design and optimization strategies need to be configured quickly and with minimal effort to arrive at effective design strategies. And, according to a recent paper, redesigns occur almost daily at major industrial institutions.

Why FIDO needs a face-lift

The original FIDO framework, designed to illustrate the viability of heterogeneous high-performance computing, lacks the capability to be used as design tool-kit. In real life, an optimi-

zation specialist needs to be able to quickly reconfigure his/her system by rewiring the components. In spite of all the good things one could say about FIDO, it needs to be pointed out that:

- (a) To simulate new optimization strategies using FIDO requires considerable rewrite of code, especially at the driver level.
- (b) No capability exists in FIDO to pick and choose from a suite of discipline codes at execution time.
- (c) Problem setup using a graphical user interface is not operational for the current version of FIDO. Monitoring capabilities can be charitably defined as primitive; post-processing of results is rather painful.
- (d) The addition of design variables or files to the problem database is a labyrinthine process that few have ventured into and fewer still have mastered.
- (e) The infrastructure is not very robust and portability has been an overwhelming challenge to the FIDO team. The recent upgrade from SunOS to the Solaris operating system caused a considerable disruption to CSC's task schedules; side-effects visited upon those entrusted with resuscitating FIDO included premature graying and heart palpitations.
- (f) The current infrastructure does not provide for the corroborative environment critical to fostering speedy MDO development.
- (g) The infrastructure was developed using procedural, not object-oriented, principles making the framework hard to comprehend, modify, and extend; framework design and organization is undocumented and was performed on-the-fly by the project designer. The project designer is also no longer associated with the FIDO effort.
- (h) Use of distributed object standards facilitate heterogeneous computing; such technology was not feasible when FIDO was conceived four years ago.
- (i) To alleviate concerns of technology transfer, components such as communications library, database, monitoring, and display were developed in-house. Quite often, application engineers have been forced to utilize considerable labor in debugging the system and tracking inconsistencies and race conditions. Consequently, focus on engineering aspects of the optimization problem has been diluted.

In FIDO's favor, considerable work was completed in the last couple of years on incorporating discipline codes and demon-

strating multidisciplinary optimization. FIDO's analysis capabilities need to be preserved; it's the infrastructure that needs to be re-engineered.

So the real question is *not*: Do we fix FIDO (painful pun not intended) or replace it? The question that needs to be addressed is: How can we upgrade FIDO's infrastructure by incorporating characteristics that MDO enthusiasts desire?

Which brings us to the central issue. What exactly are we looking for in a framework? Or better yet, *what characteristics of a framework facilitate optimization methodologies?* The sections that follow lay out the issues that need to be addressed for typical optimization situations. The characteristics of an "ideal" framework for such MDO processes are explored.

Typical Optimization Scenario

Multidisciplinary analysis and optimization is a process that is dominated by collaboration. A typical scenario is one whereby a group of people working at different sites collaborate on the design of an aircraft. Management of design process requires that modifications to one part of the design be communicated to all (or a subset) of those involved in the design process.

How it really works!

In theory, this collaboration sounds like a reasonable process. But it has all the hallmarks of a bureaucrat's fantasy. An excellent example of a real-life optimization process is one that was enunciated by a recent visitor to NASA Langley from McDonnell-Douglas Corporation, Joe Giesing.

Giesing talked about the design and optimization process that industry typically follows. The industry norm goes like this: A project leader holds a weekly meeting with a group of people.

Each member of this team usually possesses substantial expertise in one or more disciplines. To optimize, say, a wing design, these experts journey to a central location where they huddle around a table and indulge in technical deliberations. Design changes to the wing are bandied about until a consensus is reached on what design perturbation is most beneficial. The project leader goes back to his workstation, incorporates this change, and runs through numerous optimization cycles to arrive at a new design for the wing. This new design is presented at the next weekly (or monthly) meeting to gauge its acceptability.

If the design process is complex, as it usually is for airplane components, modifications to the design are anything but simple to instrument. The optimization procedure may have to be rewired; the effort might take a week to accomplish, maybe more. This process will involve engineers modifying various codes to reconfigure execution scripts, schedule file transfers, validate sections of code in a stand-alone mode, and run through the many steps required to engage multiple analysis segments running on disparate computer resources.

Constructive Collaboration

The above scenario reflects the reality of collaborative efforts, and by extension, is representative of most multidisciplinary optimization projects. MDO projects inevitably spans buildings, branches, divisions, research centers, states, and countries.

An ideal collaborative environment will have the ability to bring all these disparate resources to one place. It would allow every person involved in the collaborative effort to show the others the impact of design changes in any discipline. This setting would illustrate the relative importance of most discipline changes on the overall design. All the players would have an

input into how the design system was configured. They would be able to see the impact of their design changes in a matter of minutes or hours. One can easily visualize the complete design being completed in a few hours rather than the customary months it now consumes.

Traditionally, multidisciplinary optimization methodologies have necessitated the construction of specialized frameworks that are built by experts. These systems are characterized by inflexible procedures, requirement of specialized knowledge, hard-coded work-flow characteristics, forbidding entry ports, and requirement of specialized computer science know-how required for its operation. Some existing frameworks (I shall refrain from naming names) are so convoluted that even engineering experts are scared off by the barbed wire around them. These barriers are a distinct hindrance to the development of efficient MDO strategies. So, in these times, what constitutes a good framework?

Our Ideal Framework

Let us remind ourselves just *who* these frameworks are designed to help. Any organization that requires multiple disciplines (or processes) to be configured quickly in various proscribed manners requires the functionality described below. The order in which these requirements are presented is a reflection on the relative importance they hold in useful, user-friendly frameworks.

Tool-box Property

The framework should incorporate a tool-box functionality for engineering analysis codes, both legacy and modern. It should allow an user to incorporate codes such as COMET, CFL3D, and WINGDES with relative ease. Access to source code is beneficial in situations wherein a code might be required to run

on user-specified platforms. Most engineers don't care about how the framework incorporates these various codes, but only that they are available for use.

The tool-box property should really be extended to all aspects of the framework. In particular, components such as the Graphical User Interface (GUI), database access, and communication infrastructure that cradles this framework require this characteristic. This will enable the framework to be relocated with minimal effort to any plausible site such as the Internet, a workstation cluster, a heterogeneous network of computers, a parallel supercomputer, or to any user-selected combinational configuration.

Object-Oriented

The various disciplines should be able to be hooked up in whatever fashion the user desires. This is a loaded request and certain important implications follow. One is that each *program unit* should be encapsulated. A program unit is the lowest level of stand-alone executable that an user wants to define. Encapsulation implies that an user should not care how the structural displacements are obtained but that an object advertised to provide this capability, does.

The definition of program unit implies that macro-objects, which are collections of program units, can also be configured and defined by the user. Thus, a hierarchy of object definitions is possible and indeed, desirable. This is akin to stepping up to the MDO fast-food counter and saying, "gimme an aeroelastic loop object with ISAAC and ELAPS and the KSOPT optimizer on the side".

Object Interface Specs

The generation of interface specifications for each program unit (aka object) is critical. Interface specs are the handshakes

that enable the various program units to be civil (and communicate) with each other. They are handlers advertised by each program unit which allow an user to connect up many program units without having to mess with the gory details of each unit implementation. The connections between the various program units are best done in a visual programming environment. Our ideal framework should also take an additional step and refuse to wire up incompatible program units or objects.

Efficient Object Linking

The user should be able to select the code that a particular process will use from among a group of like-objects. That is, the user should be able to wire up an aeroelastic loop that allows easy substitution of ISAAC for WINGDES. A similar functionality should be available for model fidelity (i.e grid resolution) selection.

The selection of a number of program units forms a *process*; our ideal framework should facilitate process generation. For example, an user should be able to look at a skeleton model or a wave-drag deck and query the framework:

1. Generate a volume grid with $(n_i * n_j * n_k)$ points.
2. Show me how the model looks.
3. Use this model with the aerodynamics code of my choice.
4. Finally, show me the resulting pressure distribution.

If the user is satisfied with the results of his exercise, he should be able to encapsulate each subprocess or all these steps into a *persistent* process. A persistent object is one which retains its properties after it has been used and the system or framework has been disbanded. This process or macro-object would have behavior and interface specifications that could be used by other processes (or objects).

The generation and cataloging of these well defined macro-objects would facilitate optimization procedures. Any user

would be able to pick up a well-tested and validated macro-objects (for example, a low-fidelity weight estimation module) and use it in his/her analyses.

Good Graphical Interfaces

To be able to perform efficient object linking, the Graphical User Interface (GUI) should be well-designed. As an example, the user can design a process flow diagram with a generic aerodynamic button; when pressed, this button will display a list of aerodynamics codes that are available to the user. The user can select an aerodynamics code with the click (or double-click) of a button.

Object design and development tools will allow the construction of flow charts describing the design processes. They can visually describe the interconnection between objects, and relationship among macro-objects. Such visual programming tools dramatically reduce the time it takes for a novice user to become productive.

The capability to display results, i.e. post-processing, is very essential for frameworks that are employed for engineering analysis. The ideal framework would facilitate results display by configuring display drivers. A file-type can be used to trigger the execution of appropriate display modules, or, a particular icon can be associated with a file type. The MIME-type specifications that all modern Web browsers employ is an ideal candidate. For instance, an user can display the stress contours using PATRAN by double-clicking on the “stress.patran” file. This capability will be very similar to the Microsoft Windows file icons; each file has an icon which, when double-clicked, invokes a default display program.

High-Performance Computing

Utilizing all the specialized resources that are available to the MDO engineer is a top priority. In complex design processes (such as those associated with airplanes) an analyst will need to utilize high-fidelity analysis codes. These codes are usually optimized to run on specific high-performance platforms such as the IBM SP2. Since a design process might need to have such analysis codes invoked many times, it is imperative that the framework facilitate such analysis capability. An ideal framework should facilitate information exchange among the disparate computational resources that an analyst might want to employ for that design.

Most research centers and industries have at their disposal networked workstations, parallel supercomputers, a multitude of workstations based on various operating systems, as well as a collection of personal computers (PCs). Depending on a user's needs, the framework should rope in all of the requested resources.

The framework should be extensible in order to support current and future high-performance computing platforms. Today, the term high-performance computing implies use of parallel supercomputers; tomorrow, it might mean a specific subset of the millions of computers that populate the Internet; the day after, that's anybody's guess. Our ideal framework would allow specialized programs to operate on tailored architectures.

Patterns of communication allowed by the framework must be unrestricted. An application using distributed databases might need peer-to-peer connections; so will the framework that contains a segment whose work-load is distributed among multiple processors. Frameworks that are locked into one message-passing paradigm or another, or those that use specialized data

transfer protocols will have a hard time adapting to technological changes.

Plug-and-Play Capabilities

One way to facilitate rapid reconfiguration of the design process is to be able to link modules at run time. Each object or macro-object should be compiled separately; these executable objects are not globally linked together before execution. The way these modules communicate is through object communication mechanisms such as Object Requests Brokers (ORBs). If a particular module needs to be debugged, this methodology will allow that module to be separately modified, tested, and put back into the execution loop.

A particular advantage of using this object technology is that these modules or objects can be reside at disparate locations. To quote John Stands, a technology specialist at Sprint, “Distributed objects are prime time. If you’re not thinking distributed objects, you’re behind the times.”

Multimedia Collaborativity

Since we are primarily interested in multidisciplinary optimization problems, the entire design process needs to reap the benefits of collaboration. This collaboration should begin with the way each object is assembled into the design framework, and permeate the entire design process. This indicates that the framework should be able to incorporate the latest multi-user environments, whiteboard technologies, and interactive audio, video and textual capability.

Two or more people should be able to collaborate on how to wire up an optimization process. The wiring should be done such that this exercise is visible to multiple users; each user sitting at his/her computer can see the connections that this process designer is performing in real time.

Typical MDO teams comprise of experts. Each expert can contribute his or her technical expertise to the design process by being involved in the layout of program units. An expert in one field can also review details of archived layouts to configure other disciplines into a new optimization procedure. Each member of an MDO team would be able to communicate their thoughts in a highly collaborative audiovisual environment. Advances in audio, video, and conferencing capabilities would be incorporated into this ideal framework to make it seem as if all participants were there in the same room and sketching their ideas out on the same piece of paper.

This sort of collaboration will ensure an efficient design that is approved by the whole group and contributed by all. Experts in all fields will ensure that the overall design process conforms to all discipline constraints. The old adage of “Too many cooks spoil the broth” is not applicable to MDO applications; contributions from multiple experts facilitates good design.

Functional Interactivity

Each object will have predefined properties and default initial values for its variables. So when an optimization process is completed or a macro-object fully assembled, each object displays its variables to the user. At this stage, the user can modify variable values of any object, but not its behavior.

Our idealized framework might also have predefined rules of behavior. By that, we indicate the capability of the framework to evaluate an user’s request and react in a manner as to enhance the integrity of the design process. An example: If an user entered an estimate for the structural weight of the airplane, the framework should query the user with, “Do you want me to run FLOPS to check that weight?” This would help remind the user to verify his/her objects and macro-objects before embarking on an expensive MDO analysis. Also, some measure of artificial intelligence can be embedded into the

framework--for instance, an user would be warned if he/she entered an unrealistic structural weight of 10 lbs for a hyper-sonic vehicle!

The framework's interfaces should be intuitive to the user. By that we mean that an user would not be required to lug around mammoth user-guides or have to attend a week-long course taught by the vendor. A good example is the ease with which users of Microsoft Word come up to speed using those 3 million lines of code! Granted, a complex MDO process will not be as simplistic as a word-processing program. But, intuitive interfaces, built-in learning kits, and readily accessible help files will make MDO users very productive.

Configuration Archival

Once the user completes his wiring diagram or problem layout, he/she should be able save this configuration. Also, during the process of constructing this program layout, the user should have a "journal" facility. Journal utilities ensure that all the process decisions are documented and specific design steps taken are recorded. Such journal files allow an user to generate a new optimization procedure quickly by making minor modifications to an existing one, speed up generation of program modules, and help new users understand the whys and hows of an existing optimization layout. A journal file also helps to archive the knowledge of senior designers; expert systems can be evolved to emulate the design steps authored by the experts. Also, journal files help enormously when unexpected events, such as computer system crashes, occur at inopportune moments in the design process.

Resource Allocation

The previous sections dealt with issues concerning the construction of optimization layouts. The next step is to imple-

ment the computational procedure for a specific design problem. Each distinct program unit associated with this problem can be assigned to a specific computer. If a program segment does not reside on the processor where the framework is initiated, then the various pieces of information required for that segment need to be shipped to its destination. Information that needs to be resident on the remote computers include source files, data files, makefiles, scripts, etc.

Each process (or macro-object or object) will be assigned a default computer type as well as a list of choices where they could be exercised. For instance, the code *WINGDES* will have *SUN4* as its default architecture; machines such as *clyde*, and *cmb48* will be listed as its likely destination. If the user changes the architecture option to *rs6k*, the framework will select among a list of the available machines of that particular architecture. A knowledgeable user can also override the framework's defaults and customize his resource environment.

Any user-defined macro-object will keep a running tally of its resource counter as objects are incorporated into it. Macro-objects will use their previously defined values if they have already been through an optimization process. If not, these smart objects will add up all the resources that are required by their component objects and then suggest an appropriate computational destination. The framework will offer its recommendations; but it assumes that, when assigning resources, a human is smarter than a package of hardware and software components.

Problem Inputs & Outputs

Each process (macro-object or object) can inform the user about the output values it can provide. The user can elect to use default values, or pick and choose the variables that need to be displayed during each cycle. Files that need to be saved at the end of each cycle, those that need to be saved at the end of the

optimization procedure, and those that can be discarded each cycle can be tagged by file markers. The user can also select the processes that need to output log files, debug output, etc.

Global variables such as the number of optimization cycles, number of design variables, and convergence criteria for the optimization problem should be defined via easily manageable user interfaces.

Time-line of Frameworks

MDO frameworks have varied life-lines. A person performing a rigorous optimization study will need to keep his framework intact for many days/weeks. Others might need to use a collection of computational resources for only a few hours. This will be the case when an analyst needs to run through a number of volume grids to ensure that his aerodynamics segment provides consistent pressure loads. Whatever the life-line, when a particular framework finishes its defined goals, it should be disbanded. With the journal file capability mentioned earlier, regenerating the process flow diagrams is not arduous.

Sharing of Program Resources

The ideal framework should be clever when it comes down to managing its resources. By resources we mean objects that are encapsulated or “wrapped” computer programs. Let us take an extreme example: One un-natural afternoon, *every* researcher in the MDO organization is driven by some strange impulse to perform detailed optimization studies on an airplane wing. About a dozen researchers set up their optimization processes on the same day, at the same time, using the same analysis codes. Can our ideal framework handle the resource crunch?

As mentioned earlier, our ideal framework would have each program unit encapsulated as an object. Two users may not be allowed access to the same object simultaneously if one modi-

fies variables accessed by the other. All the dozen or so frameworks that are invoked on this unusual day should have access to *current, validated and centrally-maintained* versions of all analysis programs. These free-floating objects would service all the users by either explicit time-slicing or using the concept of threads.

Consider the notion of sharing analysis codes: Allowing each researcher to have his or her own versions of each analysis program can cause problems such as (a) lack of standardization--when one code is modified or enhanced all versions of this code must be updated, (b) the space taken up by multiple version of a program is wasteful if no differences exist among the versions, and (c) not all researchers will need all program objects (i.e., analysis program executables) stored permanently in their space.

Sharing an analysis capability among many users does put a dent in overall execution time. But, realistically speaking, how often does it happen that multiple users need to access the same computational resource, on the same day, at the same time? Anyhow, if such an unlikely situation is found to recur, popular objects can be duplicated to handle extreme service demands.

Float like a Butterfly, Fly like a Bee

The previous section talked about how the analysis programs are objects that are shared by the users. The framework, on the other hand, will be resident on the user's workstation. This implies that the framework should not be a heavy monolithic executable but a collection of services. These services are like a skeleton that will allow all sorts of modules to be plugged in. One user can start with this framework and plug in his favorite analysis modules, display programs, GUIs, and communication units. Only a small set of functionalities will be perma-

nently ensconced in the framework. Each framework user can tailor his or her version to possess the desired capabilities.

The overhead associated with framework maintenance is simplified since only the core functionality needs to be maintained. Upgrades to the framework, changes in the core software components, changes in operating systems, and new functionality are issues that are handled by the framework administrators. Such issues don't interest the average user; our ideal framework allows the engineering analyst to focus on the design and optimization issues.

Platform Independence

Going hand-in-hand with the notion of a localized framework, is the need for platform independence. It would be convenient for an user to invoke the analysis or optimization programs from his/her workstation/PC/Xterm and not be forced to use alien computers or operating systems. The look and feel of the framework should be the same no matter which operating system the user uses or where the user decides to initiate his/her computations.

Intangible Criteria

Since we live in a world that is quite not Utopia, a framework that satisfies all the criteria described above does not exist, at least not at this time. The task herein is how to compare the various frameworks that exist to decide which ones to focus on. Such a comparison process is complicated by the concerns listed below.

Ugly Ducking Or Swan Or In-Between

Some of the primary issues that the framework evaluators must grapple with are:

(a) Framework A trumpets almost all the capabilities an MDO infrastructure requires. How much of that is hype (or wishful thinking)? How much is legitimate capability?

(b) Framework B has a lot of capabilities. It is commercial and some of the infrastructure is proprietary. If NASA wanted to extend the functionality of the framework, would that be permitted? Would the product be supported? Would the learning curve be short or long? Would those involved in adding functionality have to learn an obscure or specialized language construct? Can NASA hand out the framework to industry once that framework demonstrates its functionality?

(c) Framework C is not as complete as A or B. But it has the potential to be extended since it is built on a strong and flexible foundation. The framework, though incomplete, has excellent technology going for it. How much of a factor should its extensibility be in the selection process?

(d) Framework D has a lot of functionality but the framework is rather large. Each user will need a complete copy of the framework on his workstation. Would each installation be priced separately? How many versions of the framework can run at the same time? Will the framework limit access to those using PCs, Macs, etc.?

(e) Framework E is a skeleton infrastructure but looks like it can easily assimilate the best pieces from a lot of other frameworks? Does NASA want to spend the resources needed to develop a “Utopian” framework? Can resources be dedicated to such a project that will promote NASA’s technical prowess and provide industry with the most appropriate MDO tools?

(f) Which framework has an infrastructure that enables easy transfer of all the discipline-related work that was accomplished in FIDO?

Candidate Frameworks

Preliminary evaluations indicated that a few of the frameworks that currently exist had characteristics that were worth exploring. This evaluation phase involved perusing brochures, publications, documentation, and usage manuals obtained from proponents of these frameworks. Wherever possible, efforts were made to contact users of the system to poll their experiences with that particular framework. No effort was expended in actually exercising the capabilities of any framework.

An initial survey netted a large number of application frameworks that possessed some or many of the characteristics described in the previous pages. Since it was impractical to evaluate a score of infrastructures, the list of frameworks was pared down to five that seemed particularly applicable for the MDO applications that FIDO addressed. A concise description of the candidate frameworks as well as their primary positive and negative attributes are given in the following sections.

TACTICS

TACTICS (Tri-service Advanced Countermeasures and Threats Integrated Combat Simulation) is a framework developed by TASC of Reading Massachusetts, and funded by the U.S. Army Tank Automotive Command in Warren, Michigan. To quote the TACTICS Vision and Overview Manual: *The TACTICS infrastructure is a set of interconnectivity services intended to provide the application-oriented user flexibility and ease-of-use in rapidly configuring simulations composed of either custom-developed or legacy codes.* Talk about a mouthful of the right buzz-words!

TACTICS promises to enable an user to construct engineering simulations using *plug-and-play* objects that can interact across distributed computing platforms. The framework is intended to facilitate the easy rewiring of simulations and enable users to plug in their favorite analysis, display, and post-processing modules.

POSITIVE ASPECTS

- ✓ The framework is not a set of simulation programs explicitly tied together but a collection of interconnectivity communication services written in C++. These services include communication, binding, interfaces and data probes.
- ✓ The framework is based on object-oriented principles and application of object software design standards and tools. These enable easy extensions and modifications to the framework.
- ✓ Communications between the modules and objects is effected using an industry-standard protocol - CORBA.
- ✓ CORBA allows distributed computing across heterogeneous platforms. This is a critical requirement for MDO applications.
- ✓ Legacy codes can be easily converted to work in this framework using interface wrappers and registering them as CORBA-compliant objects.
- ✓ Adding new simulation capability requires minimal effort; for example, the classes used for wrapping WINGDES can be extended by inheritance concepts to wrap ISAAC.
- ✓ The framework is light; each user can customize his or her version with appropriate codes. The plug-and-play architecture and abstraction layers will allow new display, post-processing, and database modules to be plugged in as and when they become available.

- ✓ The framework has the capability to utilize graphical module connector programs to generate code required for defining the problem flow.
- ✓ The executables of analysis codes are treated as objects. This allows only one module of the program to be debugged and then linked during runtime. The entire framework does not have to be compiled when one analysis module needs to be modified.
- ✓ NASA can obtain the TACTICS framework for free. This includes the infrastructure source code, graphics libraries, and some service modules.
- ✓ COTS (Commercial Off-The-Shelf) products are used liberally in this framework. Using current tools available in the market reduces development time and software maintenance efforts significantly. Today's commercial arena is marked by rapid developments and torrid competition, resulting in COTS products that are relatively inexpensive and platform-independent.

NEGATIVE ASPECTS

- ✗ CORBA is a new standard and its capabilities are still evolving; TACTICS uses an Object Request Broker (ORB) that is undergoing revisions and upgrades.
- ✗ To utilize this framework, NASA will need to download a freeware CORBA implementation or purchase commercial CORBA products, such as IONA's ORB.
- ✗ An army transportation engineering group at Fort Eustis is working with Bell Helicopters to use TACTICS or a subset of TACTICS for helicopter simulation. Communications with this group indicate that not all the advertised capabilities in TACTICS are working as advertised.

✘ A collaborative environment can be developed with TACTICS as the base. The primary drawback would be the complications involved in coordinating the action of WWW-based objects (Java applets) with C++ objects using object brokers.

MIDAS

MIDAS (Multidisciplinary Integrated Design Assistant for Spacecraft) is a distributed environment developed at the Jet Propulsion Lab in Pasadena, California. It is specifically designed for multidisciplinary analysis. MIDAS has a good visual programming front-end, includes commercial and native design tools resident on heterogeneous computational platforms, and allows modules written in FORTRAN or C. Algorithms for multidisciplinary optimization of a system can be investigated by allowing the master module or scheduler to be wired up in proscribed ways. Communications between the various modules is effected through the popular Parallel Virtual Machine (PVM); a CORBA-compliant version of MIDAS is said to be forthcoming.

The basic design paradigm of MIDAS is multiple tools running on a network where the results are passed from one tool to another under the control of a master design program. The master can also extract data, build input files, and start execution of tools. The focus of MIDAS is not to work on automated ways to do single point design; MIDAS has a grander agenda--capturing the knowledge in a designer's head which allows him/her to come up with feasible designs. The logic is that once that is accomplished, then a computer could be taught how to go through the same reasoning steps hundreds of times to pick optimal candidates.

POSITIVE ASPECTS

✓ The framework has a good visual programming interface--a natural graphical way for engineers to design their project and perform the interconnections between modules.

- ✓ MIDAS allows legacy codes, commercial codes such as NASTRAN, and applications such as spreadsheets to be used in the design process.
- ✓ It allows the user to utilize heterogeneous computing resources such as personal computers, a network of workstations, and parallel supercomputers.
- ✓ MIDAS allows multiple concurrent execution of analysis codes on available computational resources.
- ✓ Allows an user to look at pre-existing design paradigms that he or she can modify to speed up the definition of a new optimization procedure.
- ✓ Allows an user to track the progress of the optimization graphically. This will help the user terminate the process if the design was progressing in the “wrong” direction.
- ✓ All the designers work concurrently, and analysis performed by one designer uses latest designs of peer designers.

NEGATIVE ASPECTS

- ✗ While MIDAS uses C++, the documents perused do not indicate how object-oriented the framework is. The concern is that the classes used for the design should facilitate the extension of MIDAS. Classes that enable easy selection of different analysis codes, and those that can be extended to provide visualization capabilities are examples that come to mind.
- ✗ Components in MIDAS seem to be tightly coupled to the framework. This will be a drawback if the user wants to invoke his favorite database, monitoring, post-processing, and/or communication modules.
- ✗ Program resources are not shared by multiple users; each user has his/her own copy of all the analysis modules and utilities.

✘ The concept of plug-and-play modules is not well developed in the current version of MIDAS; no doubt, the CORBA-compliant version will alleviate this situation.

✘ It is not clear as to how a collaborative environment exists in MIDAS. Enabling problem setup, monitoring capabilities, and results display such that a team of people are involved in all these processes is not implemented.

iSIGHT

iSIGHT is a framework developed by Engineous Software, Inc., located in Morrisville, North Carolina. It is touted as the leading MDO management software that is commercially available. iSIGHT has been applied to the design of many products and employs several state-of-the-art optimization techniques to produce optimal designs. iSIGHT's architecture promises to “*support an extensible, customizable toolkit approach for tailoring the environment to an organization's needs.*”

iSIGHT includes an impressive collection of modules that can be used for pre- and post-processing and monitoring capability.

POSITIVE ASPECTS

✓ The framework allows users to integrate their own or third-party analysis codes, visualization and monitoring tools, system services, and user interfaces; source codes for the analysis programs are not required.

✓ iSIGHT manages the exchange of design data between the simulation programs automatically. This facilitates integration of analysis codes from multiple disciplines.

✓ Has toolkits that have sophisticated controllers to manage the coordination of tasks within and between toolkits. An ana-

lyst can explore more design options and perform parametric trade-off studies more effectively.

✓ Seminal strength of iSIGHT seems to be its suite of optimization techniques. Modules that implement Concurrent Sub-Space Optimization (CSSO) and Collaborative Optimization (CO) are planned in the next release of iSIGHT. Optimization can be run either in interactive or batch mode.

✓ Can utilize high-performance number-crunchers such as parallel supercomputers and workstations.

✓ Uses expert knowledge to give the user suggestions and warnings about their problem formulation. Checks for data sufficiency for constraints, design variables, or program inputs.

✓ Has a plan for incorporating a facility for representing an engineer's knowledge and inferences on that knowledge to effectively control movement through the design space.

✓ Has a data management toolkit which provides a replay capability, allows functional approximations, a high-speed look-up facility to avoid rerunning already available states, and keep a record of which files were output for each design state.

✓ Includes the designer interface module **foreSIGHT** which is a convenient way for engineers to provide problem formulation information and control of simulation codes.

✓ Includes **overSIGHT**, a module that provides various output display charts to monitor variables, constraints, or objectives during an optimization run. The **hindSIGHT** module is a data manipulation and visualization environment to analyze design data from a completed run.

✓ The **farSIGHT** module, which enables applications to be coupled using point-and-click technology, is in the works and expected to be delivered by March 1997 in the next version of iSIGHT.

NEGATIVE ASPECTS

- ✘ Support for heterogeneous computing appears thin. The framework proponents claim to have used a proprietary package (HLA) that overlays CORBA for inter-processor communications. Parallel supercomputers can be utilized, but in an unsophisticated manner.
- ✘ No source code will be available, even for the optimization modules that currently exists in iSIGHT.
- ✘ iSIGHT appears to be a monolithic program that will reside on each user's workstation. Multiple users may not be able to run iSIGHT (with one license) but they can run "subsets" of the system with a single manager controlling the principal iSIGHT invocation.
- ✘ Support for a collaborative environment seems limited. It appears that multiple users cannot access problem display nor contribute to the design of the aircraft or the coupling of the applications.

INFOSPHERES INFRASTRUCTURE

The Infospheres Infrastructure (*II*) is a distributed system framework being developed at the California Institute of Technology in Pasadena, California. The infrastructure provides a generic object model and a variety of message-passing models. The *II* imagines future distributed systems that span the globe, utilizing every resource, be it software or hardware, on the Internet. The infrastructure calls this global distributed system the World Object Network (*WON*). In such a network, a group of objects can be brought together to perform some useful work. Such a collaborative linking is termed a "virtual" network.

The *II* project is aimed at designing information infrastructures that supports virtual organizations by exploiting advances in

network-centric tools. The vision of *II* is to empower an object on the Internet such that these objects can monitor the states of processes, communicate with other objects, and respond to events. Such “active” objects can be collared to perform collaborative tasks such as those for multidisciplinary design and optimization.

Some of the unique features of *II* are: (a) the design of a supportive infrastructure that includes people and resources from various organizations, (b) development of algorithms that scale with the number of available resources, (c) study collaborative structures and the systematic design of collaborative applications, (d) designing the framework such that it adapts quickly and extends flexibly to incorporate new resources, (e) designing objects such that they yield resources when they are not required and acquire resources when dealing with an event, and (f) building interfaces to objects such that security issues such as user authentication are handled.

The *II* project is slated to work on cutting-edge issues such as scalability, management of resources, varied task durations and message delays, object interfaces that exploit commercial technologies, access control to objects, servicing a collection of objects involved in a session, and graphical interfaces to represent a collection of objects that need to be connected together.

The prototypes in *II* use Java and the Internet (IP addresses and sockets), and support for other languages and communication structures are planned in future versions. The intent of the *II* system is aimed at letting non-programmers create powerful personalized distributed programs.

POSITIVE ASPECTS

✓ The *II* is written using Java, a fully object-oriented programming language that include a great deal of predefined application interfaces. The use of Java simplifies a lot of the regular programming tasks enabling faster program development.

- ✓ The entire source code as well as extensive documentation is freely available and can be downloaded from the Internet.
- ✓ The cutting-edge technology utilized in the *II* enables a framework based on *II* to include the latest commercial innovations such as whiteboards, collaborative environments, universal accessor methods, Web-based client-server technologies, and database connectivity.
- ✓ Multiple people can access the Internet using Web technology to set up, monitor, and chart the progress of an optimization procedure.
- ✓ The *II* is a light-weight framework that can reside on all popular platforms. Connections to the C and FORTRAN executables can be done using Java native method calls.
- ✓ Any functionality or method that is available on the Internet can be utilized by this framework. The Universal Resource Locator (URL) is used to identify and utilize any object or executable that is required for the optimization procedure.
- ✓ The infrastructure allows users to integrate their own or third-party analysis codes, visualization and monitoring tools, system services, and user interfaces.
- ✓ Using the concept of mailboxes, *II* manages the exchange of design data between the executing programs using reliable TCP/IP protocols and sockets.
- ✓ All the processes or objects will need operating system services check-pointing, synchronization, etc. To handle these requirements, *II* has a library of operating system services that these processes can access. This library should be able to handle problems associated with communication, network, and delays.

NEGATIVE ASPECTS

- ✘ The infrastructure is a government-funded, university-based research project; this implies that the framework functionality will ebb and flow continuously.
- ✘ A good deal of development effort will have to be undertaken to add MDO functionality to the existing *II* framework
- ✘ The framework may need to incorporate a CORBA-compliant Object Request Broker (*ORB*) to enable high-performance heterogeneous computing. Freeware Java-based *ORBs* are available and may be used to fill this void.

DAKOTA

The Design Analysis Kit for OpTimizAtion (DAKOTA) is being developed at Sandia National Laboratories. DAKOTA uses the C++ language to develop a tool-kit for advanced optimization studies. A primary thrust of the effort is to use the newest optimization algorithms within a framework that facilitates engineering analyses. The DAKOTA system also enables an user to manage his/her system resources efficiently by incorporating multiple levels of parallel processing, supplying multiple system models, and facilitating multiple communication protocols.

POSITIVE ASPECTS

- ✓ DAKOTA is written in C++ and claims to be strongly object-oriented; this enables extensions to the framework and make it flexible for software development.
- ✓ The use of interfaces enables the analysis codes to be encapsulated. Application Interfaces are designed such that multiple codes and codes of different fidelity can be interfaced with the discipline drivers.

- ✓ The DAKOTA framework has a library of optimization modules, such as DOT, NPSOL, OPT++ and SGOPT, which could be useful for optimization studies.
- ✓ Incorporating legacy codes is facilitated through the use of Application Interfaces. All types of code (except those that require proprietary interfaces) can be installed in DAKOTA.
- ✓ The entire source code as well as extensive documentation is available to NASA.





NEGATIVE ASPECTS

- ✗ DAKOTA aspires to be a problem-solving environment, but the emphasis is on optimization strategies. MDO applications require interfaces that will have to be generated.
- ✗ DAKOTA has been run on an IBM SP2 and past emphasis seems to be on high-performance computing not heterogeneous distributed computing. The use of communication protocols for heterogeneous applications such as CORBA is planned for the coming months.
- ✗ No formal methodology for handling multidisciplinary couplings exist in DAKOTA. The strategy is to use discipline drivers to facilitate transfer of appropriate information.
- ✗ DAKOTA does not possess a visual programming interface or tool-kit to set up the MDO problem or to define the connections between the analysis codes.
- ✗ Post-processing capabilities are very limited. In the near future, the use of Netscape or WWW technologies are planned to provide visualization capability.

Eyeball-to-Eyeball Comparison

One of the primary concerns in any comparison between the frameworks is a question of how to represent each framework's strengths and weaknesses. The previous sections listed the positive and negatives associated with each framework, but a bird's-eye view of the frameworks' capabilities will be beneficial and easier to assimilate. The symbols in Table 1 indicate how well the frameworks match up to our "Utopian" framework. In the table that follows, Table 2, each framework is

TABLE 1. Symbols used for ranking frameworks

Symbol	Level of Support
	Excellent
	Good
	Fair
	Poor/None

ranked using these symbols. The desirable properties list in this table are those that were discussed in the first few sections of this document.

The ranking that follow are based on the author's conclusions of that support level stated in the brochures or projected based on the framework's technology base.

TABLE 2.

Comparison of Frameworks (TAC -> TACTICS; iSIG -> iSIGHT; MID -> MIDAS, II -> Infospheres Infrastructure; DAK -> DAKOTA)

Desirable Properties	TAC	iSIG	MID	II	DAK
Tool-box Capability	●	●	●	◐	◐
Object-Oriented	●	◐	●	●	●
Interface Specs	●	●	◐	●	●
Efficient Object Linking	●	◐	◐	◐	◐
Good Graphical Interfaces	◐	◐	●	○	○
High-Performance Computing	◐	○	◐	◐	○
Plug and Play Capabilities	●	○	◐	●	◐
Multimedia Collaborativity	○	○	◐	●	○
Functional Interactivity	◐	●	○	◐	◐
Configurational Archival	○	◐	●	◐	○
Resource Allocation	◐	○	●	◐	○

TABLE 2.

Comparison of Frameworks (TAC -> TACTICS; iSIG -> iSIGHT; MID -> MIDAS, II -> Infospheres Infrastructure; DAK -> DAKOTA)

Desirable Properties	TAC	iSIG	MID	II	DAK
Problem Inputs & Outputs	●	●	◐	◐	○
Time-line of frameworks	◐	◐	◐	●	◐
Sharing program resources	◐	○	○	●	○
Framework Lightness	●	○	○	●	◐
Platform Independence	○	◐	○	●	○

CSC's Recommendations

In looking at our comparisons, a few things become apparent. Aside from annoying spots that persist even when one looks away, one can see that (a) most of the frameworks have their strong and weak suites, (b) the award of ranking probably reflect the author's biases (try as he might to stay objective), and (c) if the criteria for framework selection was modified, the number of *excellent's* awarded to the various frameworks might change appreciably.

Given the caveats mentioned above, it is still worthwhile to offer our recommendations on which frameworks deserve a closer look.

Recommendation 1: If NASA has the resources to invest in a framework that will use cutting-edge technology, leverage the commercial software industry, possess the dynamics to evolve

and grow with technological innovations, and provide the sort of collaborative technology that the aircraft industry will welcome with open arms, CSC's choice would be the Infospheres Infrastructure.

The informed reader can quibble (quite justifiably) that the Infospheres Infrastructure (which received a substantial number of good and excellent grades) is a research tool. The infrastructure, in its current state, is admittedly primitive, but the technology that it leverages has extremely good potential for MDO frameworks. *II* takes full advantage of an existing heterogeneous, distributed, ubiquitous, multidisciplinary, multi-faceted, multimedia infrastructure: the Internet. With an adequate infusion of resources, a version of FIDO 2.1 that exploits *II* could be available by the end of 1997.

Recommendation 2: The next best alternative would be to borrow heavily from MIDAS for the new FIDO framework. In fact, select features of TACTICS can be incorporated into this MIDAS-based framework to develop an infrastructure that will satisfy many of the items in the desirables list. Resource requirements for this effort will be lower than the effort proposed in the previous paragraph. This framework can be tailored to work on heterogeneous platforms using CORBA, utilize high-performance computing, assimilate FIDO discipline drivers and analysis codes, and utilize display facilities in TACTICS and MIDAS. It may even be possible to borrow from *II* some aspects of the collaborative environment infrastructure. This is possible since Java objects can talk to C++ objects using CORBA. An **optimistic** projection is that the FIDO 2.1 problem could be incorporated in this new FIDO framework by the September 1997.

Thus, CSC's primary recommendations are that some effort be expended in exploring the utilization of *II*, *TACTICS*, and *MIDAS*. If the time-line associated with these projections is not in line with NASA's schedules or milestones, CSC's prescription is the recommendation below.

Recommendation 3: Of all the frameworks reviewed, *iSIGHT* appears to be the most complete for MDO applications. With *iSIGHT* one can start evaluating various optimization strategies within a couple of months. The framework also includes a lot of pre-processing and post-processing packages that an analyst can employ to setup, monitor and verify optimization strategies. For the short term, using *iSIGHT* makes a lot of sense; MDO researchers would be able to hit the ground running and be productive quite rapidly. For the long term, *iSIGHT* may not be quite what the MDO community requires. *iSIGHT* is a commercial package which implies that it may not have the flexibility a research environment requires. Such an infrastructure discourages interactive collaboration and inhibits technology transfer. As mentioned before, *iSIGHT*'s primary strength is all the optimization options that it offers; NASA Langley (and MDOB) is focused on developing new optimization strategies, not performing design studies. Thus, our recommendation to embrace *iSIGHT* is tinged with reluctance.

Concluding Remarks

The primary emphasis of this effort was to develop some guidelines for what an “ideal” MDO framework would possess. The frameworks we investigated could then be graded based on these guidelines. These grades have a subjective component associated with them. In particular, we had to make a judicious hypothesis as to how open each framework is to assimilate emerging technologies. The framework that NASA selects will need to be productive for the next few years; adaptability to innovations in engineering and technology is important.

It must be mentioned here that frameworks such as the Adaptive Modeling Language (*AML*, developed by TechnoSoft Inc.) and *IMAGE* (developed at Georgia Tech) were initially consid-

ered for evaluation, before being cast by the wayside. These frameworks may fall in the “worth a second look” column under changed considerations. There may be other priorities driving the framework selection such as restrictions in costs and resources. Also, there may be those who will take issue with the contents of the desirable characteristics list and their relative importance. It should be emphasized that this document is a preliminary comparison between candidate frameworks that show promise for MDO applications.

There is a great deal of effort underway to demonstrate high-performance computing in MDO applications. High-performance computing is obviously aimed at making sure engineering applications are optimized to run at blinding speeds. But, another subtle component of high-performance computing in MDO is how fast can an analyst set up his optimization problem and how easily he/she can leverage available computing sources. Cost-effective designs are obtained by reducing the time it takes to set up the methodology to run multiple analysis codes. High-performance computing, by this definition, allows an engineer to accelerate his/her design by providing the user with enabling environments and appropriate tools.

The combined clout of NASA’s High Performance Computing and Communications Program (HPCCP) and Multidisciplinary Optimization Branch (MDOB) could facilitate the generation of a new framework. NASA Langley can lead the way in developing a MDO framework that exploits all the leading-edge technologies of the day. Such a framework will benefit NASA’s primary partner, the commercial aircraft industry, and energize MDO applications in other industries. Such a bold effort might require an infusion of resources and personnel. But, to paraphrase PBS, **if NASA Langley doesn’t do it, who will?**

References

1. Engineous Software, Inc., "iSIGHT Technical Overview", Morrisville, North Carolina.
2. Bachinsky, S.T., Gokhman, B., Hodum, F.J., and Bailey, T.A., "TACTICS: An Infrastructure for Integrating High Fidelity Simulations," TASC Report, Reading Massachusetts, 1995.
3. Svedlow, M, "TACTICS Vision and Overview," TASC Report, 1995.
4. George, J., Peterson, J, and Southard, S, "Multidisciplinary Integrated Design Assistant for Spacecraft (MIDAS)," AIAA Paper, AIAA-95-1372, 1995.
5. Team Midas '95, "Multidisciplinary Integrated Design Assistant for Spacecrafts (MIDAS)," Jet Propulsion Laboratory, Pasadena, California, November 1995.
6. Chandy, K.M., Rifkin, A., Sivilotti, P.A.G., Mandelson, J., Richardson, M., Tanaka, W., and Weisman, L., "A World-Wide Distributed System Using Java and the Internet," Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing August 6--9, 1996, Syracuse, New York", IEEE Computer Society Press, 1996.
7. II: The Infosphere Infrastructure User's Guide, The Infospheres Group, California Institute of Technology, Pasadena, California, November 1996.
8. Eldred, M., Personal Communications to Jim Rogers, January 1997.

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704-0188	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE October 1998	3. REPORT TYPE AND DATES COVERED Contractor Report	
4. TITLE AND SUBTITLE Evaluation of Frameworks for HSCT Design Optimization			5. FUNDING NUMBERS NAS1-20431	
6. AUTHOR(S) Ramki Krishnan				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Computer Sciences Corporation 3217 North Armistead Hampton, VA 23669			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration Langley Research Center Hampton, VA 23681-2199			10. SPONSORING/MONITORING AGENCY REPORT NUMBER NASA/CR-1998-208731	
11. SUPPLEMENTARY NOTES Langley Technical Monitor: John J. Rehder				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified-Unlimited Subject Category 61 Distribution: Standard Availability: NASA CASI (301) 621-0390			12b. DISTRIBUTION CODE	
13. ABSTRACT (Maximum 200 words) This report is an evaluation of engineering frameworks that could be used to augment, supplement, or replace the existing FIDO 3.5 (Framework for Interdisciplinary Design and Optimization Version 3.5) framework. The report begins with the motivation for this effort, followed by a description of an "ideal" multidisciplinary design and optimization (MDO) framework. The discussion then turns to how each candidate framework stacks up against this ideal. This report ends with recommendations as to the "best" frameworks that should be down-selected for detailed review.				
14. SUBJECT TERMS Multidisciplinary Design Optimization; MDO; Computational Framework. Aircraft Design			15. NUMBER OF PAGES 42	
			16. PRICE CODE A03	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	