AIAA 98-4942
Computational Frameworks for Collaborative Multidisciplinary Design of Complex Systems

D. E. Acton
J. R. Olds
Space Systems Design Lab
Georgia Institute of Technology
Atlanta, GA

7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization

Sept. 2-4, 1998 / St. Louis, MO

Computational Frameworks for Collaborative Multidisciplinary Design of Complex Systems

David E. Acton *
Dr. John R. Olds †
Space Systems Design Laboratory
School of Aerospace Engineering
Georgia Institute of Technology, Atlanta, GA 30332-0150

Abstract

Significant research has been conducted into the development and deployment of analysis integration techniques. This paper discusses these research activities in the context of collaborative frameworks for conceptual space systems design. It begins by discussing the current state of aerospace systems design processes, then recommends a path toward higher efficiency in design. It continues by describing various computational technologies that have been or are being developed to achieve this goal, and concludes by presenting several demonstration problems that make use of these technologies.

Introduction

Research Context

The Space Systems Design Laboratory (SSDL) at Georgia Tech is one of three branches of the Center for Aerospace Systems Analysis (CASA). SSDL is focused primarily on developing conceptual design strategies and tools for advanced launch vehicles, satellites, and interplanetary spacecraft. The material discussed in this paper is based on research into integrated computational frameworks for complex system design. The design framework is intended to link together disciplinary analyses in a distributed, heterogeneous computing environment, and automate data exchange for the purpose of collaborative design. The research is performed within the SSDL, and therefore tends to have an emphasis on space systems design. The demonstration problems involve both launch vehicles and interplanetary spacecraft, so the analysis tools deal with space-related disciplines.

One good reason why space systems design is well suited to an integrated design framework is its inherent complexity. Figure 1 shows a typical Design Structure Matrix (DSM) for a launch vehicle design process performed in the SSDL. The first thing to notice is the relatively large number of disciplines involved when developing such a vehicle. This DSM does not even show some other disciplines that may be involved in more detailed designs, such as "Stability & Control," "Avionics," and "Operations & Facilities."

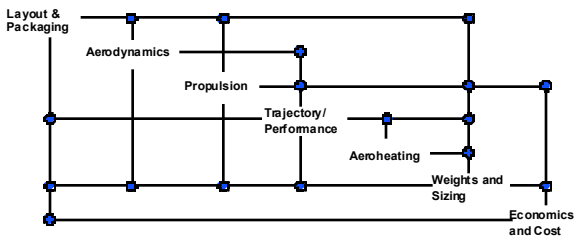

Figure 1. Typical Design Structure Matrix for an advanced launch vehicle.

The second thing to notice about the DSM is the high level of coupling between the various disciplines. Decisions that are made early in the process (for example, in "Layout & Packaging") will undoubtedly need to be modified when the results of later analyses are generated. Typically, the process is iterated until convergence of the design variables is reached. This means that the entire design process may take many iterations, each involving numerous complex analyses[1]. An integrated design framework can help organize the transfer of data between different disciplines.

The large amount of resources spent performing various analyses also encourages the application of design frameworks to aerospace-related problems. The space design community makes wide use of industry-standard *legacy codes*. These codes, whether they are command-line based FORTRAN programs, graphical

---

* *Graduate Research Assistant, School of Aerospace Engineering, Student member AIAA.*

† *Assistant Professor, School of Aerospace Engineering, Senior member AIAA.*

CAD programs, or PC-based spreadsheets using industry standard formulas, typically require a great deal of human interaction to execute. The user may need to hand-edit text input files, sit in front of a CAD terminal drawing a model and analyzing it, or edit and retrieve cell values of a spreadsheet using a mouse at a PC. If a computer-based integrated design framework can reduce the tedium of using legacy codes by performing repetitive tasks automatically, it has instantly enhanced the design process.

Conceptual Design Strategies

In today's industry, the method of designing complex engineering systems usually takes one of two forms. The *Monolithic Design Code* integrates contributing analyses as subroutines of a large synthesis "executive." Intended for single user execution, usually in the conceptual design phase, a monolithic code uses response surfaces, low-fidelity codes, and other methods to approximate the higher fidelity results that would be obtained using legacy codes.

At the opposite extreme lies the methodology of *Loosely-Integrated Analyses*. This implies heavy use of standard legacy codes, but provides for very little electronic integration of contributing analyses. Experts in each discipline manually perform each analysis, and data is exchanged between disciplines along written or verbal lines, or possibly through *ftp* of data files. This method is usually seen in large organizations, working on large, complex systems, often at the preliminary or detailed design phase.

With current methodologies, it is difficult to bring high-fidelity analyses into the conceptual design phase. Correspondingly, when the engineering system is transitioned into preliminary and detail design phases, the convenience of easy data exchange and fast cycle time is lost. Current research is directed towards developing collaborative design frameworks that bridge the gap between the existing design strategies, and thereby share the benefits of each. An optimum design scheme is one that reduces cycle time, yet keeps the fidelity and accuracy of legacy codes and disciplinary experts.

These results are accomplished by transforming the complex disciplinary analyses into "design-oriented" analyses, which are then "plugged into" a larger system executive. These *Tightly-Integrated Analyses* will not be approximations, but rather true legacy tools set up by experts. The codes are "wrapped" using shell-scripts, remote computer control, and other methods. The system executive automatically executes each contributing analysis when needed, returning a high-fidelity result that can be easily passed to the next analysis.

Thrust Areas

While the goals and benefits of a tightly-integrated design framework are obvious, the mechanisms by which to achieve it may not be. The primary problem is how to integrate and automate legacy codes and resources that were designed to be executed by humans. Issues that will be encountered include

- Designing and developing a computational architecture that will support integration and design process deployment through a system-level executive
- Difficulty in automating sometimes complex interfaces to legacy analyses
- Dealing with dissimilar computer platforms for analyses
- Extending the framework to access geographically distributed computer systems
- Demonstrating the effectiveness of integration and automation techniques on relevant sample problems

The research discussed in this paper addresses these issues and examines methods to resolve them in a tightly-integrated collaborative design framework. Four demonstration problems were constructed to showcase the integration technologies and act as "proofs of concept" for the design framework.

This is a good time to differentiate between a *design framework* and a *computational architecture*. A design framework is defined as a methodology for connecting disciplinary analyses. The framework refers not to a particular piece of software, but rather to the idea of how codes should be tied together (if at all) in a design environment. A computational architecture is defined as the enabling software environment that supports a particular framework. The architecture may be best represented by the system-level executive described above.

It is important to note that this research is not intended to develop a particular computer architecture for engineering design, but rather to examine the techniques necessary for integrating analysis tools into such an architecture, and compare existing architectures. Much work has been done in both academia and industry to construct computational architectures that support integrated design. However, what many design architectures leave out is a clear description of how to integrate a wide variety of codes.

Admittedly, the architecture designer cannot predict the complete array of analyses that a user may wish to integrate, and therefore cannot define every technique for wrapping such analyses. Rather, he supplies the "ports" for the architecture, into which the engineer "plugs" his wrapped analysis codes.

When discussing a future design framework, one must first look at the way design is done in industry now. Decades of engineering experience and tradition cannot just be wiped away, requiring a completely new way of designing spacecraft, aircraft, or any other engineering system. That would mean telling engineers to change the way they make decisions and perform design. There are too many legacy codes and legacy knowledge that must be tapped. Rather, one should look at the way design is done now, then look at how the design process can be improved upon in the future, estimate what efficiencies can be obtained, and develop ways to bridge the gap.

Architectures for the System Executive

The structure and implementation of the system-level executive is a critical issue in designing the environment. The executive defines the user interface to the design process while it is running, as well as possibly during problem setup. The executive also determines the method of data storage, data retrieval, and data exchange[2]. The effectiveness of a particular executive may be measured by its ease of use during both setup and execution, robustness to both user error and design instability, expandability to include additional analyses, and level of automation.

Two system executive implementations were investigated, including a pre-existing Unix-based single-user architecture that provides sophisticated integration and database tools, and a Web-based architecture that allows distributed, multi-platform information access and analysis execution.

Unix Interface

The architecture chosen in this category was the Intelligent Multi-disciplinary Aircraft Generation Environment (IMAGE)*. Developed by Dr. Mark Hale as part of his Ph.D. dissertation at Georgia Tech, IMAGE is a modular computing architecture based on a Form, Model, Process design theory[3]. In its current form, IMAGE consists of a graphical user interface, based on the Tk/tcl scripting language, above an object-oriented modeling system (See Figure 2). It contains a structured scheme for integrating analyses, supports communication via PVM, and can be compiled for a variety of Unix platforms.

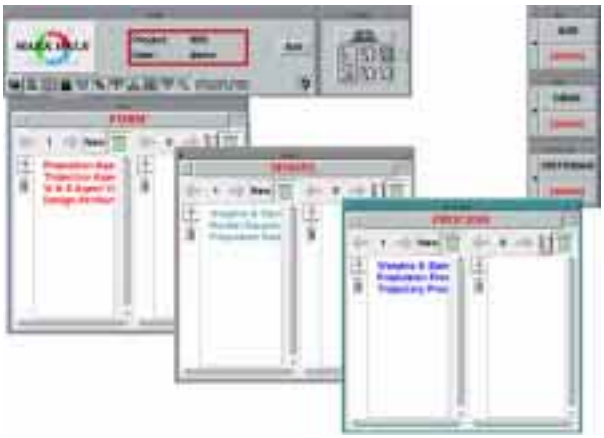

Figure 2. The IMAGE integrated design architecture.

It is useful to describe the primary elements of IMAGE in order to develop a context within which one may later discuss its effectiveness for certain problems. IMAGE is based on the notion of design schemas. A schema is a collection of various objects that all have a common basis. A Form schema will likely contain a collection of variables related to a particular subsystem. An example from the aerospace field would be the collection of variables related to a propulsion system. A Model schema will contain objects which define a model of a particular phenomenon. This phenomenon may be the aerodynamic behavior of a vehicle in atmospheric flight. In order to model this behavior, it is simulated via a computer code. The Model schema will describe the computer code and how it is executed. A Process schema will contain objects which link particular models to particular design variables. It is no good to have an aerodynamic analysis if it is applied to propulsion design variables. The Process schema provides the coordination mechanism.

Once all schemas and processes are defined, the design problem can be executed. During execution, interpreters are started as agents are called by IMAGE. When the problem is completed (or even during execution), variable values within the Form schemas may be examined. A very useful feature of IMAGE is its ability to retain past values of each design variable. Instead of keeping only the most recent number,

* The name IMAGE is a bit of a misnomer since the architecture really contains nothing that makes it inherently appropriate for aircraft design versus any other sort of system design.

output file, or picture, IMAGE stores past versions, and records each with the date, user, and project with which it is associated. For numerical values, the progression of design variables may be plotted to help in visualizing the convergence (or divergence) of a design.

Web Interface

The World Wide Web (WWW or Web) provides a natural architecture on which to build distributed computational frameworks. The combination of its Internet heritage, simple and consistent data request formats, and customizable levels of security makes the Web appealing to someone planning to run widely distributed codes.

The Web uses the HyperText Transport Protocol (HTTP) and Uniform Resource Locators (URLs) to pass information between various locations. The way URLs are formatted, any page of information, or any Web-accessible computer program may be accessed via a simple one-line text string. The page or program may return content ranging from simple text data to graphical images, plots, and sound files. This rich content is passed directly back to the user via the HTTP protocol, and displayed in his or her Web browser. Another benefit of the distributed Web environment is that no real distinction is made between a Web server located upstairs and one located across the country. Contributing analyses may be easily ported from one location to another.

Web servers may be run from a multitude of platforms, including Unix machines, Macintoshes, and Windows-based PCs. This makes the problem of handling different computing environments easier to deal with. If the system executive is web-accessible, it is easy for design team members to monitor the design process by simply loading up a page in their web browsers.

The research presented in this paper investigates the use of the Web architecture to support remote execution and automation of several legacy codes. The Web architecture provides a familiar and attractive interface between the designer and the knowledge-base. As the world's computer users become increasingly interested in using Web-based services, it seems natural to extend engineering design into this environment. However, the field of Web-programming is a relatively new one, and therefore is still developing. Even with its obvious benefits, the Web architecture falls short in some areas. The Web was originally designed simply as a way to display information, and therefore provides limited means for the user to actually interact with that information. Only recently, with the advent of the Java and JavaScript programming languages, have users been able to manipulate the information they see in their browsers. Even these languages, however, have a long way to go before they can compete with the highly developed graphical interfaces provided in the Unix, Macintosh, and Windows development environments.

For most Web servers, the only database existing behind the interface is the Unix or PC filesystem containing the Web pages. It is certainly possible to connect say an object-oriented database behind a Web server, with Java and C++ helping to provide the required hooks. However, an exercise such as this would require as much expertise as connecting the same type of database behind a graphical Unix/X-Windows interface.

Another major problem for a Web-based design environment is the inability to define an automatable process model. The Web was designed for human interaction, so it usually involves manually clicking buttons and selecting hyperlinks to move forward in a process. Unlike the Unix architecture, which can spawn processes when preceding processes are finished, a Web server will typically just show you the results page, and wait.

Integrating Legacy Codes

Legacy resources, while providing high-fidelity results, are often not design-oriented. Design-oriented resources must be suitable for automated and repetitive use, provide consistent results, accept wide variance in input parameters, and often allow for visualization of output. Many legacy tools, such as CAD systems and spreadsheets are hard to automate. Many companies are designing into their software capabilities that allow for automating some processes, and this is a step in the right direction. What about those programs that are not receiving that sort of attention? Should designers restrict themselves to those codes that are already automatable at the expense of loosing fidelity or learning a new program? Probably not. So techniques must be developed to automate a wider variety of tools.

Even for a relatively simple command-line program, automation takes some skill. For example, the space design community often uses the trajectory analysis code POST (Program to Optimize Simulated Trajectories) to analyze the ascent and reentry performance of conceptual launch vehicles[4]. The output of this code includes a single large text file containing the numerical trajectory results for the run.

The numbers that the designer is interested in are located within the file, or derivable from numbers in the file, but extraction of the numbers usually requires loading the file in a text editor and manually picking out the required values. When POST is automated, this tedious task is performed completely by the computer.

Wrapping Unix-based codes

POST serves as a good example of a typical command-line based Unix legacy tool. The program remains today as it was originally written decades ago, a FORTRAN program run from a Unix command-line, taking a single namelist-formatted text input file (commonly referred to as the *input deck*), and generating several text output files. An expert carefully sets up the input deck for a nominal vehicle design.

When the trajectory portion of the vehicle design is to be converged with other analyses, or the converged vehicle is to be optimized, a few key design parameters in the input deck are changed. Some design variables may not be just a single number, but rather tables of values. In this case, the entire table is substituted into the input deck via a C-style "#include" statement.

After execution, the designer extracts certain key values from the output files that help him judge the trajectory performance. Data output from POST that is in tabular form may be plotted to gain visual insight into the vehicle's performance.

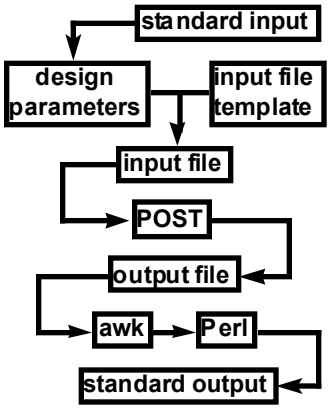

Figure 3. Execution process for an automated Unix command-line analysis.

An "agent wrapper" was designed for the POST program that automates most of the procedures described above. The wrapper takes the form of a Unix Korn Shell (ksh) script, executed at the command line with several arguments. The ksh script is convenient since it makes use of Unix commands and syntax that most users are already familiar with. Drawbacks include limited capabilities, especially with math functions and code reuse. Figure 3 shows a schematic of the execution process. An input deck template is populated with the input variables provided on the command-line (initial weight, reference area, etc.). The resulting file is given a ".inp" extension and submitted to POST on an SGI machine. Once POST has finished, the output file is parsed for the last instances of certain key variables. These values are used to calculate the desired output quantities (mass ratio, mixture ratio, etc.), which are then sent to the user terminal or piped to an external file.

This same procedure may be used for a variety of input/output file analysis codes. For example, the Georgia Tech-developed RBCC propulsion code SCCREAM[5] is wrapped in a manner similar to POST. In SCCREAM's case, however, the script is written in Perl, and the input files are generated in full by the script instead of by a template.

One problem that arises when automating a program such as POST is that errors during runtime are difficult to catch. POST is notorious for its ambiguous run failures. If a vehicle does not achieve orbit, the user is given little indication of why not, and the calculated mass and mixture ratio may be meaningless. The wrapping script itself has essentially no intelligence on how to deal with problems that arise. Research that is being conducted on expert systems at SSDL may help to alleviate this problem in the future.

Dissimilar Computer Platforms

Legacy resources may exist on a range of different computer platforms, thereby complicating the issue of integration and data exchange. POST is typically executed on Unix platforms, including SGI, Sun, and HP workstations. Propulsion and aerodynamics codes are also typically run in the Unix environment. The SSDL also uses spreadsheet based tools, including Weights & Sizing, and Economics & Cost. These run as Microsoft Excel files on a Macintosh personal computer.

It is important to respect the freedom that engineers have to develop analysis tools for their platform of choice. Whatever platform works best for their particular applications should be the one used, and integrators must accept this fact and work with it. The most efficient route (albeit after the integration hurdles have been surmounted) is to retain the analysis tools in their native environment. It has therefore

been a major thrust of this research to develop techniques for remotely executing tools that reside on PC platforms, from within a Unix environment.

Wrapping PC-based codes

Many of the mechanisms for communicating between Unix codes have been established over the years. However, the problem of integrating a personal computer, whether a Macintosh or Windows-based machine, presents a much larger challenge. An example of a Macintosh-based analysis tool is the Cost And Business Analysis Module (CABAM), developed in the SSDL. This Microsoft Excel spreadsheet takes a description of a particular launch system, along with assumptions about the launch market, cash handling, and production and operations systems. It provides estimates for vehicle development costs, recurring costs, cash-flow, and rate of return on investment, among other data[6].

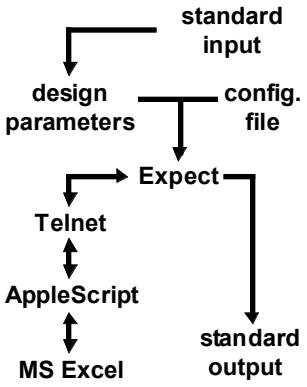

Figure 4. Execution process for an automated Macintosh-based analysis.

A wrapping script was developed to automate, and ultimately integrate the CABAM spreadsheet into a design environment. In order to provide the communications link between the Unix environment and the Macintosh operating system, two key pieces of pre-existing software were used. Peter's Scripting Daemon (a "free-ware" program available on the web[7]) provides a simple Telnet protocol server to allow a user to connect to the Macintosh. Once connected, the user may issue commands that execute AppleScript procedures. The AppleScript scripts are pre-written, so a user can sit at a Unix terminal and interact with any number of programs on the Macintosh. In this example, AppleScript is used to open Microsoft Excel with the CABAM spreadsheet, change 15 cells corresponding to vehicle component weights, read the resulting estimate of internal rate of return (IRR), then close the spreadsheet.

The procedure described above is fine when a user is sitting at the Unix terminal issuing the AppleScript commands, but what about automating the process. Telnet is a program designed for user interaction, so a scripting language called "Expect" was utilized to automate the Telnet session. Expect simulates user interaction by "expecting" certain characters to be sent back to the terminal, then "sending" other characters back. It can be used to automate "ftp" sessions, and a host of other Unix programs that rely on user keystrokes.

The general process for automating an Excel-based tool on a Macintosh is shown in Figure 4.

Widely Distributed Resources

In many cases legacy resources may not all be accessible locally. A particular code may reside on a system in one part of the country, while the design team is situated in another. This exposes the possibility of a *distributed design team*. If the design framework does not allow for distributed access to resource setup and analysis results, then the users are back to the problems of a loosely-integrated environment.

As discussed in the section regarding Web-based architectures, the Web provides a natural facility for accessing remotely located data and programs. There must also be a mechanisms in the Unix environment that allow for such remote access. This is demonstrated In the RBCC IMAGE problem to be discussed later, where one analysis tool (an Excel spreadsheet) was transported to the demonstration site and installed on a computer there. The Unix-based system executive, IMAGE, was still running off of a Sun workstation back at Georgia Tech, several hundred miles away. However, after simple changes to the wrapping script, IMAGE was able to make an agent call to the remote spreadsheet.

Developing web-based interfaces

The Web interfaces developed as a part of this research essentially build on top of the scripting techniques described in the previous two sections. These scripting techniques were originally designed to be used from the command-line, but now it makes sense to create a web interface to the tools so that other users from the launch vehicle design community can make use of them.

An example of an analysis tool that has a Web front-end applied to it is the SSDL Weights & Sizing Excel spreadsheet tool. This spreadsheet is used to

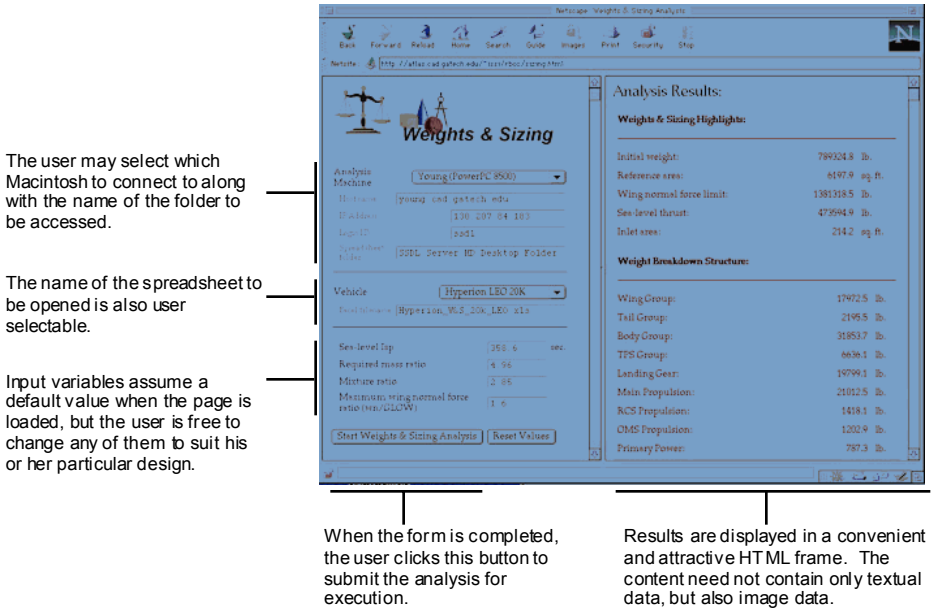

Figure 5. Web-interface to the Weights & Sizing analysis.

calculate the various component weights of a conceptual launch vehicle. It uses a combination of physics-based and empirically-based calculations, and relies on information from a variety of different disciplines including Trajectory, Propulsion, and Configuration & Layout. Since the weights spreadsheet resides on a Macintosh computer, it was wrapped in a manner similar to CABAM, using the Expect and AppleScript languages.

In order to transition such a wrapper to a Web-based interface, an HTML input form was generated (See Figure 5). This form provides the user with pull-down menus and input boxes to enter the required inputs for the analysis. The user may select which of several Macintosh machines to connect to, as well as the name of the file to be loaded. Behind the input form lies some relatively simple JavaScript code that sets certain default values and performs range checking on the user inputs. When the user submits the HTML form by clicking the "Submit" button, a Perl CGI script parses through the variable string generated by the HTML form, picking out the relevant input values. The Perl script writes out the appropriate AppleScript commands to temporary files. Perl then executes a modified version of the Expect Weights & Sizing wrapper, which in turn spawns a Telnet process to the Macintosh to perform the AppleScript commands. When the Expect script closes the Telnet connection and sends output results to the user, it sends them embedded in HTML code directly to the right frame of the browser window. In the case of weights & sizing, the output is simply a series of numbers defining the weight-breakdown of the vehicle, but this by no means limits the types of output that may be displayed. Some analyses may output such items as image files containing plots, and these are easily displayed in a Web browser window.

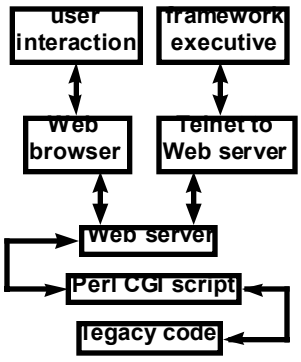

Figure 6. User interaction and framework integration of a Web-based analysis.

The HTML form/JavaScript interface method was used to wrap four important analyses in the SSDL. These include the trajectory code POST, the propulsion code SCCREAM, the economics & cost spreadsheet CABAM, and the weights & sizing spreadsheet. Figure 6 shows the general scheme in which a stand-alone code may be wrapped into a Web-based resource for use by a single user or a Web-based system executive.

Demonstration Problems

Of course, the research that has been performed in automation and integration techniques will be of no good unless it's proven that they aid in the process of engineering design. To that end, several demonstration problems were set up that make use of the techniques described above. Table 1 summarizes the four problems, and each is described in more detail below.

Simple SSTO Demonstration (Unix)

In order to show the capability of IMAGE to automate a design process, a simple all rocket SSTO design was implemented in the architecture. The problem consisted of three contributing analyses as shown in the DSM in Figure 7. Because of the absence of feedback links in this particular process, the codes were executed in sequence with no iteration between them. The purpose of this exercise was not to show wrapping capabilities, but rather the integration abilities of IMAGE. Therefore, the disciplines were represented by simple algebraic equations rather than by legacy codes or spreadsheets. This simple exercise was the stepping-stone for the much more complicated demonstrations that follow.

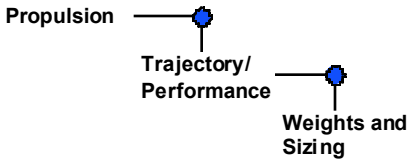

Figure 7. Design Structure Matrix for the simple all rocket SSTO demonstration.

RBCC Demonstration (Unix)

The IMAGE architecture was the basis of a four-code demonstration of the design process for a reusable launch vehicle based on a Rocket-Based Combined-Cycle (RBCC) engine. The vehicle chosen was an SSTO vision vehicle currently being studied in the SSDL named *Hyperion*. The four disciplines involved are shown in the DSM in Figure 8. The corresponding analysis tools included SCCREAM (Propulsion), POST (Trajectory/Performance), CABAM (Economics & Cost), and SSDL's Hyperion Weights & Sizing spreadsheet. The Layout/Packaging and Aerodynamics analyses were performed off-line, and the results used via photographic scaling in the other tools. The purpose of the demonstration was to show how agents can be integrated in the IMAGE environment, automatically executed according to a specified design process, and iterated to convergence without any user interaction.

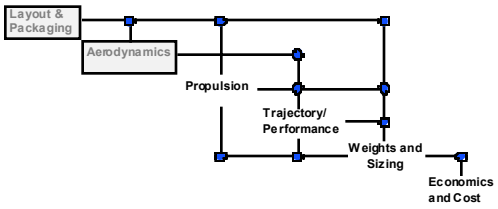

Figure 8. Design Structure Matrix for the Unix- and Web-based RBCC SSTO demonstrations.

This exercise was conducted as part of a cooperative study between Georgia Tech and International Space Systems Incorporated (ISSI), located in Huntsville, Alabama. The study was funded through NASA's Marshall Space Flight Center.

RBCC Demonstration (Web)

The *Hyperion* RBCC demonstration described above was also transitioned into a Web-based format. The user is presented with a main selection page that allows the user to perform each of the four analyses. When the user clicks on one of the images, the corresponding analysis is spawned as a Web-based HTML form interface to the code.

Table 1. Summary of design framework demonstration problems.

Name	Executive		Disciplines	Notes
	IMAGE	Web		
All rocket SSTO	√		3 (Propulsion, Trajectory, Weights)	Used simple analysis approximations to show basic operation of IMAGE
RBCC SSTO	√		4 (Propulsion, Trajectory, Weights, Cost)	Showed automatic iteration in fully-featured environment
RBCC SSTO		√	4 (Propulsion, Trajectory, Weights, Cost)	Demonstrated code interaction, and yielded improved interfaces
Planetary Entry	√	LaRC	3 (Geometry, Aerodynamics, Trajectory)	Directly compared IMAGE and Web architectures for reentry-vehicle design

It would appear that this demonstration is simply a collection of the previously developed Web interfaces, but the key difference here is the true interaction between the analyses. When each HTML form is generated, the default input variable values are read from an external database file. When each analysis is completed, the output variable values are written to this same database file, so that subsequent analyses may use the updated numbers. In this way, all four analyses may be iterated consecutively until convergence is reached.

The major drawback to this interface is the requirement that the user be present to initiate each analysis. So in a sense, one of the major goals for a design framework has been attained, namely information exchange, but complete process autonomy has yet to be achieved.

In this RBCC Web demonstration, the IMAGE architecture was essentially replaced by a single main selection page. Of course, the Web-interface should not be compared directly with IMAGE at this stage, since it contains very few of the characteristics that would qualify it as a true architecture. After all, IMAGE is not only a mechanism for executing analyses, but rather an environment in which the entire design problem is set up. Form, Model, and Process schemas, as well as the design process, are set up within IMAGE itself. For the Web demonstration, the input forms, design variables, and data exchange were all set up by hand in the basic Unix filesystem environment. With future research into Web-based system executives, however, a more meaningful comparison could be made.

Planetary Demonstration (Unix vs. Web)

The third demonstration that was conducted compared the IMAGE architecture itself with a pre-existing Web-based design environment developed at NASA Langley Research Center (LaRC). The Integrated Design System (IDS) originated by NASA provides for the conceptual design of a planetary entry vehicle using HTML form interfaces, Perl scripting behind the scenes, and SGI executables for its three primary disciplines, Geometry, Aerodynamics, and Trajectory[8]. The DSM for this demonstration is shown in Figure 9. The IDS system is quite similar to SSDL's own RBCC Web demonstration problem in how it is executed. The user is presented with a series of pages, where he selects appropriate input parameters. Perl CGI scripts execute the Unix codes and present the output data to the screen. This data consists of 2-D and 3-D geometry descriptions, aerodynamic coefficient plots, and trajectory performance plots. The IDS system is a "once-through" design process, and does not currently allow for any sort of iteration.

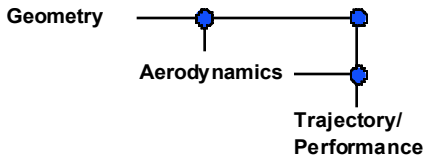

Figure 9. Design Structure Matrix for planetary entry vehicle demonstration.

As a part of SSDL's research, the IDS executable codes and CGI scripts were obtained from NASA LaRC and modified to perform within the IMAGE

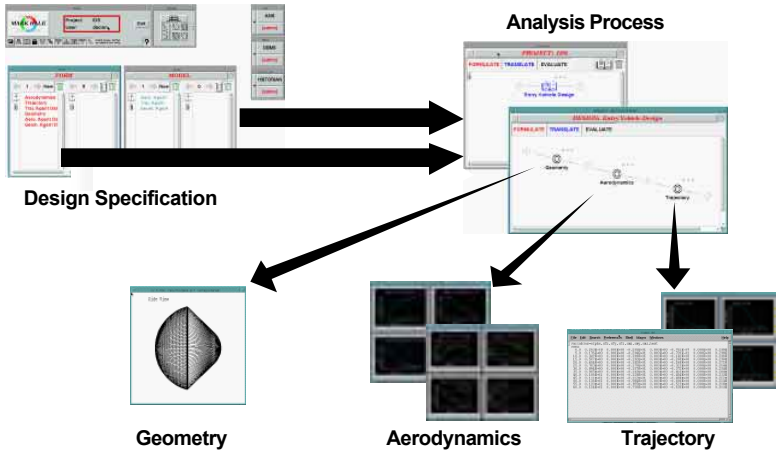

Figure 10. Planetary entry vehicle design process implemented in IMAGE.

architecture. The Form, Model, and Process schemas were constructed for the planetary entry problem, and a sequential execution process was defined (See Figure 10). The intention was to show how the exact same process could be duplicated within an alternate architecture, and compare the two approaches.

Architecture Comparisons

IMAGE and the Web-based design architectures were compared on the following basis:

- Ease-of-setup - how quickly and efficiently can one set up an engineering design problem in each architecture?
- Ease-of-use - once the problem is set up, which is more convenient to actually execute?
- Robustness of environment - how stable is the architecture when codes go awry?
- Expandability - how easy is it to change the problem once you've set it up, and how easy is it to add new agents?
- Automatability - once the problem is set up, does the user still need to be present for process execution?

IMAGE certainly outperforms the Web interface during problem setup. Its built-in object-oriented data model, while sometimes cumbersome to work with, allows the user to define variables and set up the design process using graphical tools. This is much easier than IDS' method of setting up and linking individual scripts at the Unix level.

Regarding ease-of-use, IMAGE requires little supervision if set up properly, and allows for relatively easy visualization of past results. However, the IDS and Web-based RBCC interface feels more friendly to the designer since he can actually see what is happening with his analyses. The learning curve of the IMAGE environment is quite steep, even for tasks as simple as visualizing analysis results. It is therefore not well suited to a large design team where many people require access to the design process and design data.

As a research architecture, IMAGE is not the most robust piece of software, and it requires a decent level of Unix knowledge simply to get a problem running. During execution itself, however, the system is quite reliable. Presumably, the IDS system took a good deal of time to set up properly, so most if not all bugs are worked out. Therefore the system runs quite smoothly.

When the discussion comes to expandability, both the IMAGE and IDS architectures fall short. In the current version of IMAGE, if you want to make a simple change to the design process, or add another analysis, you must completely redefine the process from the beginning. Future versions should allow easier modification. The IDS system was really not intended to be completely expandable, as it would require deep editing of Perl scripts and HTML forms to generate a new analysis interface or restructure the design process. The Web-based RBCC problem, however, provides for reasonable expandability. The input/output plugs to the flat file database are completely general, and adding a fifth analysis would only require simple modifications of that program's integration script, and the main HTML selection page.

The IMAGE architecture provides complete automatability of the design process. Once the problem is set up, the user simply clicks "Execute" and IMAGE begins running the contributing analyses, taking into account any iteration or convergence criteria. The current Web-based solutions, including the IDS demonstration, require a user to step through the design process. Neither solution is obviously superior. There may be times that user input is needed during a system design, since decisions must be made based on analysis results. This would imply that the process should not be completely automated. At the same time, however, automation capability should be *available* for those processes that do not require significant user input. Remember that one of the major benefits of an integrated design framework is the ability to execute many iterations of a design without the requirement of constant human participation.

Conclusions and Future Research

The IMAGE architecture provides a convenient and accessible architecture within which to deploy integration technologies such as those developed in this research. The detailed process model should also allow for more complicated design structure matrices to be implemented, and possibly the use of Multidisciplinary Design Optimization (MDO) techniques. Applied to a vehicle such as *Hyperion*, such a system could save countless hours of tedious code execution, and may ultimately arrive at a more optimal design.

From the feedback received regarding the SSDL's Web-based interfaces, and the results of the IDS vs. IMAGE comparison, it appears obvious that significant research should be focused on continuing the development of design architectures deployed on

the Web. The familiar and content-rich Web interface will appeal to many engineers using the design framework. It is evident, however, that problem setup is often easier in a dedicated tool such as IMAGE. In fact, there is nothing wrong with using two environments, one for problem setup and one for design process execution. Developing the problem off-line in an IMAGE sort of environment allows the use of strong graphical interface tools, while executing the process and interacting with the design on the Web provides easy access to distributed platforms, and enables distributed design teams.

Additional technologies should be developed to build the library of tools accessible in a tightly-integrated design framework. Currently, the SSDL can automate and integrate Unix-based command-line codes, and Macintosh-based analysis tools. Further study should be performed to look at communicating with Windows-based PCs, as well as complicated Unix tools such as finite-element analyses and CAD environments. As the design methodologies developed here for conceptual design begin to take hold, they will inevitably trickle down into the preliminary and detail design phases. In these phases, there will be many analyses and tools that require special wrapping techniques that have not yet been developed. Obviously, the area of computational framework research is one that has a strong future ahead of it.

Acknowledgments

This research is supported through NASA Langley Research Center contract number NCC1-229, and through a NASA Marshall Space Flight Center Phase 1 STTR in cooperation with International Space Systems, Inc.

Appreciation goes to Kimberly Steadman of the Georgia Tech Space Systems Design Laboratory for her assistance in this work.

References

[1] Olds, J. R. “System Sensitivity Analysis Applied to the Conceptual Design of a Dual-Fuel Rocket SSTO,” AIAA 94-4339, 5th AIAA / NASA / USAF / USSMO Symposium on Multidisciplinary Analysis and Optimization, Panama City, FL, September 1994.

[2] Hale, M.A., Craig, J.I., “Techniques for Integrating Computer Programs into Design Architectures,” AIAA 98-4166, Sixth AIAA / NASA / USAF / USSMO Symposium on Multidisciplinary Analysis and Optimization, Bellevue, WA, September 4-6, 1996.

[3] Hale, M.A., “An Open Computing Infrastructure that Facilitates Integrated Product and Process Development from a Decision-Based Perspective,” Doctoral Thesis, School of Aerospace Engineering, Georgia Institute of Technology, Atlanta, GA. July 1996.

[4] Olds, J. R. and Budianto, I. A. "Constant Dynamic Pressure Trajectory Simulation in POST," AIAA 98-0302, 36th Aerospace Sciences Meeting and Exhibit, Reno, NV, January 1998.

[5] Olds, J. R. and J. Bradford. "SCCREAM (Simulated Combined-Cycle Rocket Engine Analysis Module): A Conceptual RBCC Engine Design Tool," AIAA 97-2760, 33rd AIAA/ASME/SAE/ASEE Joint Propulsion Conference and Exhibit, Seattle, WA, July 1997.

[6] Lee, H. and J. R. Olds. "Integration of Cost and Business Simulation into Conceptual Launch Vehicle Design," AIAA 97-3911, 1997 Defense and Space Programs Conference and Exhibit, Huntsville, AL, September 1997.

[7] Script Daemon v1.0.2 © 1993-98 Peter Lewis. URL: “http://www.share.com/peterlewis/scriptdaemon/index.html”

[8] NASA Langley Vehicle Analysis Branch IDS System, URL: “http://vab02.larc.nasa.gov/IDS/”