AIAA 98-4743

Cross-Platform Computational Techniques for Analysis Code Integration and Optimization

J. R. Olds
K. B. Steadman
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

**For permission to copy or republish, contact the American Institute of Aeronautics and Astronautics
1801 Alexander Bell Drive, Suite 500, Reston, VA 20191**

Cross-Platform Computational Techniques for Analysis Code Integration and Optimization

Dr. John R. Olds[†]
Kimberly B. Steadman[††]
Space Systems Design Laboratory
School of Aerospace Engineering
Georgia Institute of Technology, Atlanta, GA 30332-0150

ABSTRACT

Following NASA's lead in Intelligent Synthesis Environments, advanced vehicle design communities are beginning to explore automated, distributed computing frameworks for integrating disciplinary analysis tools. These design frameworks allow collaborative design teams to take advantage of distributed expertise and existing legacy codes, while retaining some of the automation and optimization capability of monolithic synthesis tools and simple subroutines. A key capability in making these frameworks a reality will be the ability to integrate and access contributing analysis codes running on different computing platforms and in various remote locations.

This paper reports a cross-platform technique for integrating Microsoft Excel® spreadsheets into UNIX-based computing frameworks. Specifically, a combination of UNIX shell scripts, telnet connections via the Internet, and Applescript® is used to remotely execute an Excel spreadsheet hosted on a Macintosh® computer and return results to a executive program running on a UNIX workstation. Sample scripts and integration procedures are outlined. Examples are given in which the technique is used to remotely drive a launch vehicle costing spreadsheet under the control of grid search and genetic algorithm optimization techniques hosted on a UNIX workstation. Advantages and disadvantages of the present technique are discussed.

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

[††] - *Graduate Research Assistant, School of Aerospace Engineering, Student member AIAA.*

Copyright ©1998 by John R. Olds and Kimberly B. Steadman. Published by the American Institute of Aeronautics and Astronautics with permission.

BACKGROUND

NASA is taking a lead role in developing advanced engineering environments for complex aerospace systems as part of its Intelligent Synthesis Environment (ISE) and Collaborative Engineering Environment (CEE) initiatives. The goals are to reduce design cycle times, increase the fidelity of information available early in the design, and reduce life cost of the system by optimizing the initial design and avoiding costly design changes during the latter stages of the process. Key elements of these new initiative are infrastructures for distributed collaboration, rapid synthesis and simulation tools, nontraditional analysis and optimization methods, and immersive simulation environments[1,2]. Future infrastructures for distributed collaboration will take advantage of high speed networks to link people and computers at geographically distributed design centers of excellence. Product Design Centers (PDC) and Concept Design Centers (CDC) are already being setup at various NASA field centers as a precursor to this new collaborative environment.

In today's conceptual aerospace design environments, system synthesis is typically performed by monolithic synthesis codes or by manual iteration among a group of higher fidelity legacy codes operated by a team of disciplinary experts. Monolithic synthesis tools are highly integrated, standalone programs that contain a number of internal modules or subroutines for treating each disciplines. While reasonably fast, monolithic tools do not benefit from the creative input of a larger design team and typically make compromises in analysis detail. That is, they are not truly collaborative. On the other extreme, design teams operating in a manually iterated synthesis environment have the advantage of using higher fidelity legacy codes, but data exchange between individual disciplinary analyses is often slow and cumbersome. For example, finite element structural analysis and computational fluid dynamics analysis may be conducted by different engineers on different computing platforms in different states. As a result, designs are difficult to fully optimize or even completely iterate to convergence in some cases.

New computational frameworks for collaborative design promise to combine the best features of monolithic synthesis tools and manually iterated design environments. Small custom pre- and post-processing codes called ‘wrappers’ are written to automate much of the data entry and data extraction from existing (legacy) or new high fidelity disciplinary analysis tools. Once wrapped, these analysis tools become ‘agents’ to be integrated into the overall design framework. Disciplinary experts remain involved in the design process by setting up analysis tools, creating and modifying required wrappers, validating data ranges during the design process, and monitoring their own analysis results.

Computational frameworks often use scripting and telnet and remote shell commands via the Internet to allow the agents to be resident on various computing platforms and in various geographical locations (e.g. various NASA field center CDC’s and PDC’s). The design process can be managed by a World Wide Web-based or custom executive program that allows the designer to remotely execute each contributing analysis, perform multidisciplinary design optimization, view key data, and monitor the status of the design process. Taken together the analysis agents, the executive program, the framework connecting them, and associated databases and scripts are called a design architecture. In practice, the terms architecture and framework are often used interchangeably. Mature, robust design architectures and frameworks will be a key component of NASA’s Intelligent Synthesis Environment. Several research teams and private companies are actively working in this area.

INTRODUCTION

The Space Systems Design Lab (SSDL) at Georgia Tech is one component of the school’s Center for Aerospace Systems Analysis (CASA). The focus of the SSDL is advancing conceptual design of space systems — particularly reusable launch vehicles — by developing new design-oriented disciplinary analysis tools, new design processes, multidisciplinary design optimization techniques, and computational frameworks for collaborative design. Graduate students participate in various space systems design projects (design applications) and take advantage of collaborative and concurrent engineering methods.

A current research goal at SSDL is to develop and evaluate a computational, collaborative design architecture for launch vehicle design in a research environment. To parallel the needs of future design teams, the SSDL framework will contain geographically distributed agents, heterogeneous computing platforms (UNIX, Mac, Wintel), and a variety of legacy codes (custom Fortran and C codes, proprietary executables, and desktop computer-based spreadsheets). A companion paper to this one also given at this conference (reference 3) describes the current state of the SSDL architecture, includes results from applications examined to date, and summarizes a comparison between a web-based interface and a custom executive developed elsewhere at Georgia Tech called IMAGE[4]. The research reported in the present paper was a necessary precursor to many of those applications.

In the advanced space vehicle design community, many engineers rely on Microsoft Excel® spreadsheets to conduct their disciplinary analyses. Notable among the disciplines using Excel spreadsheets are cost estimation, ground operations, and mass properties estimation. These spreadsheets can be custom applications or accepted tools used throughout the disciplinary community, and they are almost always executed on the analyst’s own desktop personal computer (either Macintosh® or PC). However, disciplines such as trajectory optimization, aerodynamic analysis, and system-level optimization are often performed on UNIX workstations.

To integrate Excel spreadsheets into future collaborative design frameworks, a new wrapper must be developed and tested that enables cross-platform data exchange. This spreadsheet agent can subsequently be used as a ‘building block’ in more complex frameworks. The goal of the present research was therefore to develop the required scripts and wrappers to demonstrate cross-platform execution and optimization of a typical Microsoft Excel spreadsheet running remotely on a personal computer when driven by a UNIX workstation. Since a cross-platform technique for integrating remote Wintel-based PC codes has already been described by previous researchers[5], the present research only addressed Excel spreadsheets running on Macintosh computers. The resultant technique and scripts are described below. Two example optimization problems are also described.

APPROACH

Our approach depends on three key components, 1) Applescript to automate execution of Excel on the Macintosh, 2) 'Expect' shell scripting on the UNIX workstation to control the data exchange, and 3) Script Daemon® on the Macintosh to accept a telnet connection and route Applescript commands (figure 1). This approach makes use of existing capabilities, but does require some amount of custom programming to complete the integration. The authors do not suggest that this is the only integration approach or the best, but it has proven useful in subsequent architecture work in the SSDL.

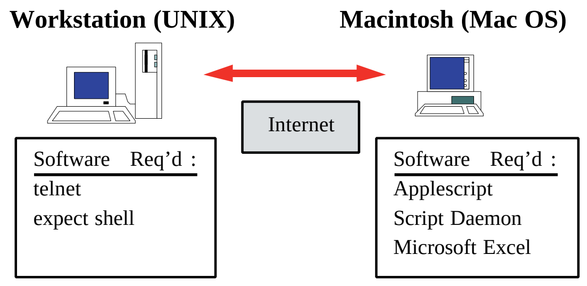

Fig. 1. Cross-platform Framework Investigated

Applescript

Applescript is a scripting language that is included with the Macintosh operating system. It allows the user to script a series of commands to control the operation of scriptable applications, including Microsoft Excel. Applescript commands exist for opening and closing a given spreadsheet, changing workbooks, entering data in individual cells, recalculating iterative results, and extracting data from resultant cells. A sample script written in Applescript is shown below. It opens Microsoft Excel, opens a spreadsheet, and returns the value of a certain cell. Application keywords appear in **bold** type.

```
tell application "Finder"
    activate
    select file "Microsoft Excel" of folder "Microsoft
      Excel 5" of folder "MS Office" of startup disk
    open selection
end tell
tell application "Microsoft Excel"
    Activate
    Open "Aldrin:Hyperion W&S 20k LEO"
    Select Range "R4C3"
    set MR to Value of ActiveCell
    set result to MR
    return result
    Close ActiveWorkbook
end tell
```

In the sample script above, the 'Select Range' command selects a specified cell of the spreadsheet. Then the '**set**' command assigns the value or number contained in that cell to a variable name specified in the script. In this case it sets a variable 'MR' to the value of cell 'R4C3' (row 4 column 3). Similar commands can be used to input values into cells.

While the syntax of Applescript is relatively easy to learn, the task of writing Applescripts is made even easier though the use a free companion utility called Script Editor. Script Editor has a 'record' function that allows the user to manually perform an operation while the Script Editor automatically generates the associated Applescript code. Once the code has been generated, it can be edited or modified as necessary in the Script Editor. Applescripts can be executed in a line-by-line interpreted fashion (inside Script Editor) or can be saved as a run-only executable mini-application independent of the Script Editor. For ordinary uses, Applescripts are resident on a local Macintosh computer and are used to simplify repetitious tasks on that same computer.

The standard set Applescript commands are fairly complete and the majority of Excel functions can be accessed via those Applescript commands. For more difficult operations, a third-party commercial product called PreFab Player® adds additional functionality to Applescript. The authors have found PreFab Player necessary to execute several functions in the newest version of Microsoft Excel (e.g. accessing Excel's built-in Solver tool or typing individual keystrokes). PreFab Player is a background application that can be activated from within Applescript to perform a variety of intricate menu, button, and keyboard commands.

Expect Shell Scripting

The 'Expect' subsystem is a free command shell available for UNIX workstations similar to the more standard k shell or c shell. It can either be a command line interface to the workstation, or can be used as a UNIX scripting language. Unlike other more common shells, Expect (as the name implies) has the built-in ability to wait for certain prompts or text strings, and then respond with a return string or character. Hence it is well suited to scripting a process that typically requires some user interface (e.g. a login process or a text based analysis tool with several prompts). A sample Expect script is given below. It logs onto a remote Macintosh using a telnet process, then sends a set of Applescript commands, and writes a result to an output file.

```
#!/usr/bin/expect –f
set infile "ssdl.inp"
source $infile
spawn –noecho telnet $mac_host

expect {Username}
send $user\n"
stty -echo
expect {Password*}
expect -timeout 1
send "$password\n"
stty –echo

expect {>*}
send "tell application \"Microsoft Excel\"\n"
send "…more Applescript commands…\n"
expect {>*}
send "$result_abbr\n"

set result "[lrange $expect_out(buffer) 1 1]"
puts $outfile_id  "$result"
send "/quit\n"
```

In this script, string variables in **$bold** are defined in the input control file, ssdl.inp. The pairs of 'expect' and 'send' commands serve as listen and respond commands throughout the remote telnet session. After the script is remotely executed on the Macintosh computer, the desired result variable is forced to be echoed to the Expect script buffer where it is captured and written to an output file. This script can be saved as a text file and executed from the command line of a UNIX workstation by typing its filename.

Script Daemon (for the Macintosh)

As discussed earlier, Applescript works well on a local Macintosh, but for the cross-platform environment envisioned in the current research, a way was needed to compose Applescript commands on the UNIX workstation and then send them to the Mac for execution. A small piece of software called [Peter's] Script Daemon does exactly that. Script Daemon is available on the Internet for public download from a variety of Macintosh users group sites. It is a small application that runs in the background on a networked Macintosh. It monitors the telnet port for connection attempts and validates the user with a username and password (the same username and password that allows owner-level access to Appleshare functions). Once a remote user is logged in, Script Daemon accepts Applescript commands and can access any scriptable application that has been loaded into memory. The following non-Applescript commands are also accepted by the Script Daemon[6].

/HELP - list the commands
/EXEC - allows you to enter a multi-line Applescript script. A period (.) typed on a line by itself signals the end of the script.
/QUIT - closes the telnet connection

In particular, the /EXEC command allows multi-line scripts to be 'collected' by Script Daemon one line at a time until a single period (.) is set on a line by itself. At that point, the entire script is executed and the outputs are returned. The result is completely remote operation of a Macintosh Excel spreadsheet from any host computer (anywhere on the Internet) that can initiate a telnet connection.

Integrating the Components

For the UNIX-Macintosh framework described here, a set of Applescript commands are first created for Excel to serve as a template for remote execution. The disciplinary expert in charge of the spreadsheet-based analysis often provides this file based on his or her normal data entry tasks. The template Applescript file is then transferred to the UNIX workstation and integrated into the Expect script. Input variable values (originally just dummy placeholders in the Applescript) are replaced with input variable names that will be dynamically replaced with new numerical input data as the design progresses. Once this preparatory work is complete, cross-platform execution of the spreadsheet is possible. The Applescript and Expect scripts are pre- and post-processing wrappers for the Excel spreadsheet. Taken together, they serve as a remote agent for disciplinary analysis.

To execute the spreadsheet agent from the UNIX workstation, the user simply creates the input file (here, ssdl.inp) with the appropriate host, username, password data as well as the current values of the design variables to be remotely entered into the spreadsheet. The Expect script is then executed from the UNIX command line. Expect opens a telnet session to the Macintosh (specifically, the Script Daemon application), sends the individual Applescript lines from the template (while substituting any new variable values), waits for Excel to perform the analysis, and collects the results that are returned. The spreadsheet results are then available in the output file on the workstation. If the number of input variables is limited, the Expect script can be modified to accept inputs directly from the command line (e.g. "runscript 400 4.5 1500 6.75"). Alternately, the Expect script can be called as an agent in a larger and more complex collaborative architecture — even from a web-based executive.

APPLICATIONS

To test and evaluate the integration methods proposed on a relevant problem, two sample frameworks were created. The disciplinary analysis modeled in both cases was the launch vehicle life cycle cost and economics discipline. An existing Excel-based analysis tool called CABAM was used as the spreadsheet model. In the first application, a modified Expect script was used to perform a full factorial grid search of four key price values in CABAM and record the results. In the second case, the CABAM agent was 'called' from within a Genetic Algorithm (GA) program on the workstation to optimize the same four price values.

Both problems required some amount of custom Expect programming on the UNIX workstation, but the CABAM spreadsheet and the Applescript part of the wrapper remained the same in both cases. It typically took even a skilled programmer 2-6 hours to configure the wrapper scripts for each application. This custom setup time is a minor drawback of the proposed integration technique.

CABAM

CABAM (Cost and Business Analysis Module) is an Excel spreadsheet tool developed at Georgia Tech to perform life cycle cost and revenue simulations for advanced launch vehicle concepts[7]. Like other cost models used in advanced design, it uses weight-based cost estimating relationships to determine much of the non-recurring cost. Facilities, operations costs, financing costs are all determined using cost estimating algorithms. To determine potential revenue, CABAM combines a user-input pricing strategy with price-elastic traffic models to determine the number of flights the vehicle flies and the revenue per flight. The cost stream and the revenue stream are combined and (after taxes are assessed) used to predict key economic indicators for the simulation like Internal Rate of Return (IRR), Net Present Value (NPV), breakeven year, and maximum debt.

CABAM requires a number of initial inputs such as vehicle component weights, complexity factors, financing rates and debt-to-equity funding ratios. Once the cost analyst as set up CABAM for a given launch vehicle, a typical task is to adjust the pricing strategy to maximize the economic performance (typically maximum IRR). Increasing the launch prices offered to the market stifles market growth and loses market share to competition. Decreasing launch prices reduces profit margin for each flight. Selecting the best prices is a discipline-level optimization problem within

CABAM. As discussed in reference 7, the gradient-based built-in Solver tool in Microsoft Excel does not perform well for globally optimizing the prices within CABAM due to the discrete and non-smooth nature of the optimization space. For example, a very small change in the price of delivering a pound of payload to orbit can result in an discontinuous change in required vehicle fleet size from 3 to 4 vehicles.

The CABAM model used in this optimization is customized for the *Argus* vehicle. *Argus* is a next generation launch vehicle with Maglifter launch assist[8]. The Maglifter is a magnetically levitated track and sled that provides 800 fps of horizontal velocity launch assist at takeoff. *Argus* has two rocket-based combined-cycle engines for main propulsion and uses lightweight materials and subsystems. *Argus* is capable of delivering 20,000 lb. to low earth orbit, or 11,100 lb. to the International Space Station orbit. It can also be configured with a crew transport module to deliver 6 passengers to the Space Station (fig. 2).

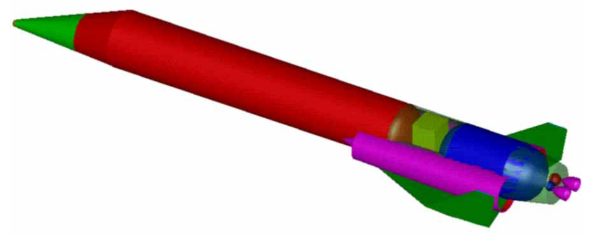

Fig. 2. Argus Launch Vehicle

The *Argus* CABAM model is set up to accept four different prices corresponding to the four different price elastic market models it is capable of addressing. They include government cargo, government passenger, commercial cargo, and commercial passenger rates. Because they all affect total flight rate, the prices are highly interrelated. Note that the commercial and U.S. government markets for cargo and passengers/tourists are treated separately because the government markets are relatively less price-elastic than future commercial markets. Prices are assumed to be per pound of cargo delivered to space station in the cargo cases or price per passenger in the case of the passenger missions. Previous trail and error methods have identified the price ranges shown in Table 1 to yield the most attractive IRR's for *Argus*.

Table 1. Price Ranges of Interest for Argus

Market	Price Range
government cargo	$1300 - 1850 ($/lb.)
government passenger	$6 - 9 M ($/passenger)
commercial cargo	$700 – 1200 ($/lb.)
commercial passenger	$0.3 - 1.3 M ($/passenger)

Grid Search

The grid search is a simple brute force optimization strategy. It evaluates every combination of the chosen design variables within a given range and resolution (i.e. a full factorial search). While expected to be very computational inefficient, the exhaustive grid search is simple to program and by searching the entire design space in a structured way, there is a high confidence in finding a true, global optimum rather than a local optimum. This is an important consideration given the known non-smooth nature of the CABAM model. Once the region of the optimum IRR is found using the grid search, the result can be locally fined tuned using Excel's built-in Solver optimizer.

Using the integration techniques discussed above, the grid search application is implemented as follows (Fig. 3). The basic Expect script structure was modified to contain four nested loops to increment the four price variables over the range of interest. At the center of the nested loop, an Applescript template is updated with the latest price variable values and sent line-by-line to a host Macintosh and CABAM (via telnet and Script Daemon). With Applescript, the new prices are entered into CABAM, the resultant IRR is calculated, and the IRR result is returned to Expect script running on the UNIX workstation. The current price variables and the IRR are written to a text file and the loop continues. The final output file will contain the results of all combinations of the four design variables. To save time, the actual telnet and user login process is placed before the nested loops in Expect so the spreadsheet is left open during the grid search. After the loops are completed, the telnet session to the Mac is closed. Depending on the speed of the Macintosh used, the process of sending four prices to the CABAM agent and returning the result may take 2 to 8 seconds. The authors are aware that this is not the most straightforward or the fastest way to implement a grid search on an Excel spreadsheet (for example, a Visual Basic® program was also written directly in Excel to perform the same task), but one of the goals of this implementation was to demonstrate the cross-platform integration and execution of the agent, and to validate it for future, more complex collaborative frameworks.

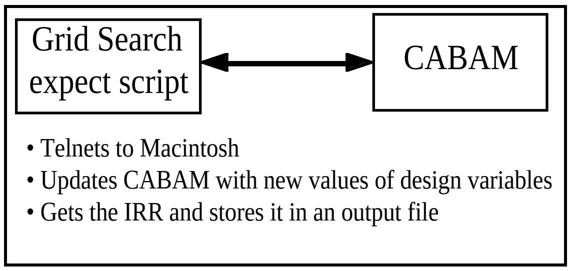
Grid Search expect script
CABAM
• Telnets to Macintosh
• Updates CABAM with new values of design variables
• Gets the IRR and stores it in an output file

Fig. 3. Grid Search with CABAM

Genetic Algorithm

The Genetic Algorithm (GA) is a more complex global optimization method that is designed to work similar to Darwin's theory of survival of the fittest. It is hypothesized that the GA will more efficiently explore the CABAM price design space and arrive at an IRR solution similar to that produced by the grid search, but in a much shorter time. GA begins with a random set of candidate locations in the design space. In this case, the candidate locations are made up of random combinations of the four market prices (each limited to its preferred range from Table 1). The initial candidates comprise a 'population' of candidates. Typical population sizes for a problem of this size is 10 – 30 different candidates. This value does not change throughout an optimization. The process of a Genetic Algorithm is to successively improve the members of the population over a number of simulated 'generations'. GA consists of three operations that are executed in order on a given population to create the new, improved population of candidate designs. The processes are reproduction, crossover, and mutation. These processes are performed in a binary space in which the true variable values are mapped to binary values and these binary values are concatenated together to form 'chromosome strings'.

Reproduction is the process where the genetic algorithm duplicates some of the 'best' chromosomes and removes some of the 'worst' chromosomes from the population. The population size will stay constant so some designs will appear more than once. During crossover, the genetic algorithm will pick two designs from the intermediate population and make two new unique designs from the two parents. This is done by splicing substrings of the parent chromosomes together to form the new chromosomes that will hopefully be better design points than either of the parents. The probability for crossover of two designs to occur is generally fairly high (otherwise the parents are passed to the next intermediate population unchanged). Mutation is a purely random process. A small number of binary digits in the chromosome string will be randomly changed to ensure that all possible combinations in the binary string will be represented. The probability for mutation to occur is usually a very low number so as to try not to change the population too much.

Once this final operation is complete, the new generation of chromosomes is mapped back to the true variable values and the CABAM agent is used to determine the best current candidate design in the population (highest IRR). Then the process is repeated for the next generation. The number of generations performed is usually predetermined (typically 5 - 20). Each generation uses approximately one call to the CABAM agent per member of the population (duplicate points are not recalculated), and due to the random nature of the GA, the entire process is usually restarted two additional times with a new initial population. So the number of times the CABAM agent is called in a GA search is approximately 3 * population size * number of generations.

Note that GA operates on discrete or discretized design variables when they are mapped to binary chromosomes. So like the grid search, the continuous price variables are converted to discrete representations over the range of interest. The resolution of the discretization process is controlled by the Nbit setting in the genetic algorithm. Nbits is the number of bits used to store each variable (i.e. there are 2^{Nbits} number of discretized values inclusively between the high and low range limits for each variable). This directly affects the number of distinct 'steps' or values a variable can attain. The GA results are very sensitive to this setting and the population size.

The genetic algorithm code used for this study is written in Fortran and runs on a UNIX machine. It was written by Peter Gage then of Stanford University[9]. The parameters for the GA are set by the user in the 'genalg.inp' file. Here the user can change variables such as the population size, number of bits, and number of generations.

Nominally, the genetic algorithm program is set up to find the minimum of an algebraic objective function. The objective function is entered into the ‘goalga.f’ input file. Our goal was to replace the algebraic objective function with a call to the CABAM agent. Changes to the original goalga.f file were made so that it issues a UNIX system call.

The new objective function system call is listed below. It writes the command 'ga a b c d' to the UNIX command line. Values for the four price variables from Table 1 are stored in the genetic algorithm as variables a, b, c, and d. So the command calls a short script named ga which in turn runs the Expect script ga.e and passes it the new values for the four price variables. ga.e subsequently accesses CABAM on the Macintosh.

```
write(cmd, *) 'ga ', a, b, c, d
call system(cmd)

open(UNIT = 23, FILE = '/tmp/goalga.out',
  STATUS = 'OLD')
read(23, *) tmp
print*, tmp
close(23)
objective = 1 - tmp
```

Note that the IRR is actually stored in the variable ‘tmp’ in this code. Since the GA is set to minimize ‘objective’, the last line converts a maximum IRR problem into a minimum objective problem for the GA (knowing that IRR is not expected to exceed 100%).

The ga.e script that updates CABAM and returns the IRR is a modified version of the Expect script listed earlier. The script takes the new values of the four design variables from the ga script, telnets to the Macintosh, updates CABAM with the new values, and returns the new IRR to the GA as the objective function value (Fig. 4). The agent includes the main file that drives the process, ga.e; a file that contains all the input parameters such as the Macintosh hostname, username, password, etc.; and a text file that ga.e reads in that contains the Applescript template commands that drive the spreadsheet. The ga.e script assumes that the correct spreadsheet is already open on the Macintosh. This was done to save execution time. However, unlike the grid search case, the telnet session was opened and closed for each CABAM function call. Future refinements in the scripts will attempt to reduce execution time and avoid unnecessary telnet calls.

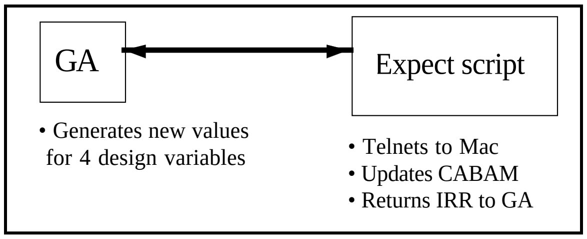

Fig. 4. Genetic Algorithm with CABAM Agent

Running the genetic algorithm with the CABAM agent calculating the objective function the command is as simple as running the GA by itself (i.e. type 'rungen' at the UNIX command line). The GA then runs through the desired number of generation and tries to optimize the four market prices. Information about optimization such as population history, best designs, and execution time are available in output files generated by the GA Fortran code.

RESULTS

The techniques discussed above were implemented on networked computers at Georgia Tech. The UNIX Expect scripts and GA code were run on a Sun SparcServer running SUNOS 5.4. The Macintosh scripts were run on a PowerMac 7200/90 running MacOS 7.6.1 (this machine is relatively slow by today’s standards). CABAM was run under Microsoft Excel version 5.0 for the Macintosh. Script Daemon version 1.0.0 was also used. The IRR results listed below correspond to an *Argus* launch vehicle configuration from November, 1997, and reflect the weights, complexity factors, and economic assumptions used by SSDL cost analysts at that time.

Table 2. Price Ranges for Grid Search

Price	Range	Increment
Gov. Cargo	$1300 - $1850	$50
Gov. Passenger	$6 - 9 M	$0.5 M
Comm. Cargo	$700 – 1200	$50
Comm. Passenger	$0.3 - 1.3 M	$0.1 M

Grid Search Results

The grid search implementation used the variable ranges and resolutions shown in Table 2 for each of the four price variables. Recall that the grid search is a full factorial and tests every combination of the four prices within the range. In this case, the method examined 12*7*11*11 = 10,164 combinations! The entire process took 1,420 minutes (nearly 24 hours) once started. This corresponds to about 8 seconds per CABAM execution.

The 10,164 results and the corresponding price inputs for each case were recorded in a large five-column text file. That file was sorted to list the results by IRR in descending order. The four highest IRR results are shown in Table 3. The highest IRR results from the grid search are slightly above 27.8% and come from the same general region of the price design space (limited by the grid search resolution). In fact, there are nearly 20 designs in the grid search that produce IRR's above 27.8%. In contrast, poor locations in this rather limited design space result in IRR's of only 23.9%. Increases in IRR of 0.1% are considered significant in this problem.

As mentioned earlier, the built-in gradient-based Solver tool in Excel can be used to fine-tune these results within a local region and typically improves the IRR by 0.1% - 0.2% above the results shown.

Table 3. Top Grid Search Results

G. Cargo	G. Pass.	C. Cargo	C. Pass.	IRR
$1650	$9 M	$800	$0.7 M	27.86%
$1650	$7 M	$800	$0.8 M	27.85%
$1650	$8 M	$800	$0.8 M	27.85%
$1700	$8 M	$800	$0.7 M	27.84%

Here, starting Solver at the top design improved the IRR to 28.06% while changing the four respective launch prices to $1642.1, $9.423 M, $799.99, and $0.725 M. It is interesting to note that starting Solver at the other three top designs always improved the IRR to nearly 28%, but no two cases resulted in the same optimized solution for the four prices — although the most dominant term, commercial cargo price, stayed very close to $799.99 in all cases. Starting the Solver at the top design was the only case that resulted in the Government passenger price increasing beyond the ranges defined for grid search. The non-unique Solver solutions underscore the non-smooth nature of this particular analysis.

While the long execution time in this application is certainly cause for some concern, the scripts and wrappers developed by this research worked well. One can only speculate at the amount of time it would take for a human analyst to type in and record 10,164 price combinations! Regarding the cross-platform execution, the authors have found by experience that the method is subject to certain pitfalls. For example, if the Internet connection to either the UNIX machine or the Macintosh is interrupted during execution, then the process will fail. Due to the relatively intensive CPU requirements, the Macintosh cannot be used for other purposes during the execution. Even background applications such as automatic email checks should be suspended. Early in our implementation, these events were found to be the source of occasional data dropouts or corrupted data being returned and written to the text file for the affected price combination. Of course, these isolated cases could always be repeated manually and typed into the IRR text output file. In general though, our experience with the grid search application was very satisfactory.

Genetic Algorithm Results

The GA implementation represents a case where the CABAM agent is integrated with a true optimization process running on a UNIX workstation. Here, one goal is to evaluate the operation of the CABAM agent within this framework. A second goal is to improve on the performance of the grid search method.

Table 4. Genetic Algorithm Results

Case	Pop Size	Nbits	Gener.	G. Cargo	G. Pass.	C. Cargo	C. Pass.	Best IRR	Time (min)
1	10	3	10	\$1535	\$8.14 M	\$843	\$0.87 M	27.69%	69
2	10	4	10	\$1593	\$9.0 M	\$800	\$0.83 M	27.83%	69
3	10	8	10	\$1701	\$8.18 M	\$814	\$0.66 M	27.80%	69
4	10	4	5	\$1593	\$9.0 M	\$800	\$0.83 M	27.83%	35
5	20	8	10	\$1645	\$8.21 M	\$829	\$1.03 M	27.81%	149

Five separate run cases were performed with the GA, with varying population size, the level of discretization (Nbits), and the number of generations performed. Each GA case was repeated three times with different random starting populations. The chance of crossover was set at 70% and the chance of mutation was set at 10%. The best results from each of these cases are shown in Table 4.

Note that Nbits creates $2^{Nbits} - 1$ resolution 'steps' for each price variable range from Table 1 (all four variables are discretized to the same number of steps in GA). For example, when Nbits = 4, the resolution on the government cargo price will be \$36.67 and the resolution on commercial cargo price will be \$33.33. This is roughly comparable to the resolution examined in the grid search and is the discretization used in cases 2 and 4. Cases 1 and 3 are relatively less and more finely resolved, respectively. As Nbits is increased, GA can address even finer resolutions over the range of interest. Increasing the resolution in the grid search would quickly become prohibitive.

While the primary goal of this research was to demonstrate the utility of the cross-platform integration procedure for a representative problem, there were several interesting GA-specific results also obtained. For example, all of the results shown in Table 4 are very close to the best IRR obtained with the grid search, but were obtained in a fraction of the time — less than 10% in some cases. This is true even though the additional telnet sessions increase the CABAM agent runtime to over 15 seconds per function call. The GA was certainly proven to be a more efficient search mechanism.

Nbits (resolution) was increased from 3 to 4 to 8 between cases 1 – 3. Surprisingly, the best result was not at the finest resolution, but was found at Nbits = 4. While the authors believe that Nbits = 3 is slightly marginal for this problem, the differences in IRR between the Nbits = 4 and 8 cases was not considered significant. Both were adequate, but higher values for Nbits are believed to result in diminishing returns.

A sensitivity test on number of generations (Gener.) in case 4 resulted in the exact same solution in only five generations that case 2 found after 10 generations! In general, a larger population size allows a smaller number of generations, but in practice a balance must be struck between the two. Setting either variable too low is unwise. In this case, the fact that the entire GA is repeated three times helps reduce the possibility of 'immature' final populations. However despite the time savings, the authors have more confidence in the settings in case 2 for a general problem of this type.

The final case, case 5, is a 'brute force' case with a high resolution and a relatively large population size. While the computational time doubles to over two hours, the results do not improve on case 2. Clearly, we have reached a point of diminishing returns. Like the grid search, GA is probably best used to perform a coarse search for the most attractive price region, to be followed up with a gradient-based method to fine-tune the results. Here, starting Excel's Solver from the price point produced in case 2 results in an improvement in IRR to 28.04%. Corresponding prices are \$1561, \$9.423 M, \$799.99, and \$0.891 M respectively. As in the grid search, the non-smooth nature of the design space is very evident. Several price combinations produce nearly the same high IRR. As before, the dominant term is commercial cargo price and is always near \$800 at the optimum.

Like the grid search implementation, the use of the proposed technique to integrate the spreadsheet into a UNIX-based optimization process was successful. In this case, the custom setup process was slightly more complex and time consuming (modifying the GA objective function to call the remote CABAM agent), but not excessive. Setup time for a skilled programmer was on the order of 4 – 6 hours.

SUMMARY

The primary goal of this research was to develop, implement, and test a cross-platform technique for integrating remote Microsoft Excel® spreadsheets used for disciplinary analysis on Macintosh personal computers with executive programs and optimizers resident on UNIX workstations. The proposed technique makes use of Expect scripting on the UNIX machine, Applescript® on the Macintosh to manipulate cells in the spreadsheets, and Script Daemon® on the Macintosh to receive Applescripts via an Internet connection (telnet). While this technique is not particularly fast, it preserves the distributed, collaborative nature thought to be important to future advanced design environments. In addition, the method has the flexibility to accommodate a range of spreadsheet-based analysis tools already in use in the advanced space vehicle design community.

To test the proposed integration techniques on a relevant problem, this study created a spreadsheet agent for advanced launch vehicle cost analysis using an existing Excel-spreadsheet (CABAM) and several custom script wrappers. Two simple cross-platform frameworks were created with the new remote agent to optimize the proposed market prices for an advanced launch vehicle. An exhaustive grid search hosted on the UNIX machine made use of the CABAM agent to perform a full-factorial search over 10,000 market price combinations. A genetic algorithm was also used to optimize the same agent, and showed efficiency gains in terms of reduced search time. Results from both methods subsequently benefited from fine-tuning within a local region using a gradient-based optimizer. Except for some limited robustness and speed concerns, the proposed cross-platform integration techniques were considered successful in these applications.

ACKNOWLEDGEMENTS

This research was partially supported under a NASA Marshall Space Flight Center STTR in cooperatation with International Space Systems, Inc. Partial support was also provided by NASA Langley Research Center under cooperative agreement number NCC1-229.

The authors wish to acknowledge the work of David Acton in helping develop key scripting techniques and procedures in support of this work. The work of graduate students Jeff Whitfield and 'Mike' Hosung Lee in developing the CABAM cost model and making it available for this research is also appreciated.

REFERENCES

1. Goldin, D. S., "Tools for Going Faster & Farther," *ASEE Prism*, September, 1998, pp. 30 – 36.

2. Noor, A. K., Venneri, S. L., Housner, J. M., and Peterson, J. C., "A Virtual Environment for Intelligent Design," *Aerospace America*, April, 1997, pp. 28 – 35.

3. Acton, D. E. and Olds, J. R. "Computational Frameworks for Collaborative Multidisciplinary Design of Complex Systems," AIAA 98-4942, 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis, MO, September 2-4, 1998.

4. Hale, M.A., Craig, J.I., "Techniques for Integrating Computer Programs into Design Architectures," AIAA 96-4166, 6th AIAA/ NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Bellevue, WA, September 4-6, 1996.

5. Moore, A. A., Braun, R. D., Powell, R. W., and Qualls, G. D., "Determination of Optimal Launch Vehicle Technology Investment Strategies During Conceptual Design," AIAA 96-4091, 6th Annual AIAA/NASA/ISSMO Symposium on Multi-disciplinary Analysis and Optimization, Bellevue, WA, September 4-6, 1996.

6. Lewis, Peter N., Script Daemon v1.0.0 Documentation, 1993.

7. Lee, H. and Olds, J. R., “Integration of Cost Modeling and Business Simulation into Conceptual Launch Vehicle Design,” AIAA 97-3911, 1997 AIAA Defense and Space Programs Conference & Exhibit, Huntsville, AL, September 23-25, 1997.

8. Olds, J. R. and Bellini, P. X., "Argus, a Highly Reusable SSTO Rocket-Based Combined Cycle Launch Vehicle with Maglifter Launch Assist," AIAA 98-1557, AIAA 8th International Space Planes and Hypersonic Systems and Technologies Conference, Norfolk, VA, April 1998.

9. Gage, P. and Kroo, I., “A Role for Genetic Algorithms in a Preliminary Design Environment," AIAA 93-3933, AIAA Aircraft Design, Systems and Operations Meeting, Monterey, CA, August 1993.