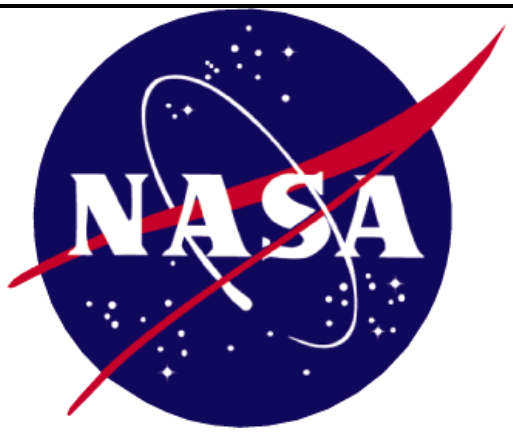
PROJECT SUMMARY NASA TASK ORDER 5 FINAL REPORT	
TASK NO.: **NNL11AB71T**	CONTRACT NO.: NNL09AA10B
PROJECT TITLE: Propulsion related module development and vehicle integration	
COMPANY: TechnoSoft Inc. info@technosoft.com 513-985-9877	ADDRESS: 11180 Reed Hartman Highway Cincinnati OH 45242
SUMMARY: This report documents the work performed during the period from May 2011 – October 2012 on the Integrated Design and Engineering Analysis (IDEA) environment. IDEA is a collaborative environment based on an object-oriented, multidisciplinary, distributed framework using the Adaptive Modeling Language (AML). This report will focus on describing the work done in the areas of: 1. Integrating propulsion data (turbines, rockets, and scramjets) in the system, and using the data to perform trajectory analysis. 2. Developing a parametric packaging strategy for a hypersonic air breathing vehicles allowing for tank resizing when multiple fuels and/or oxidizer are part of the configuration. 3. Vehicle scaling and closure strategies.	
SUBMITTED BY: Hilmi N. Kamhawi	Date: October 31, 2012

Integrated Design Engineering Analysis (IDEA) environment
Propulsion related module development and integration

Contract Number: NNL09AA10B

Report Prepared: October 2012

Prepared by:
TechnoSoft Inc.
11180 Reed Hartman Highway
Cincinnati OH 45242
info@technosoft.com
513-985-9877

This contract is sponsored by:
Vehicle Analysis Branch, MS 451
NASA Langley Research Center
Hampton, Virginia 23681-2199

Table of Contents

Introduction

The Integrated Design and Engineering Analysis (IDEA) environment is a collaborative environment based on an object-oriented, multidisciplinary, distributed architecture using the Adaptive Modeling Language (AML) as the underlying framework. IDEA has a number of analytic modules that support the design and analysis of advanced rocket and airbreathing propulsion-based vehicles. Figure 1 illustrates one such concept, a Two Stage to Orbit (TSTO) configuration with a hypersonic airbreathing first stage and rocket-based second stage. This TSTO concept has served as the initial focus for IDEA development. This report will describe continued improvements to IDEA the areas of:

1. Integration of propulsion data: Data for turbines, rockets, and scramjets were extracted from the appropriate sources and used by the trajectory module to evaluate the performance of the vehicle.
2. Turbine flowpath geometric representation: Parametric geometry was developed to represent the lowspeed flowpath. The user has access to a number of parameters to control the shape of the low speed flow path and placement within the vehicle. Geometry is connected to turbine data sheets for off the shelf engines, using dimension and mass information where available.
3. Closure methodology: A set of algorithms for closing rocket-based and hypersonic air-breathing based vehicles were developed and implemented.
4. Closure execution: A number of closure runs were executed to verify the validity of the developed methodologies and to quantify closure process times.

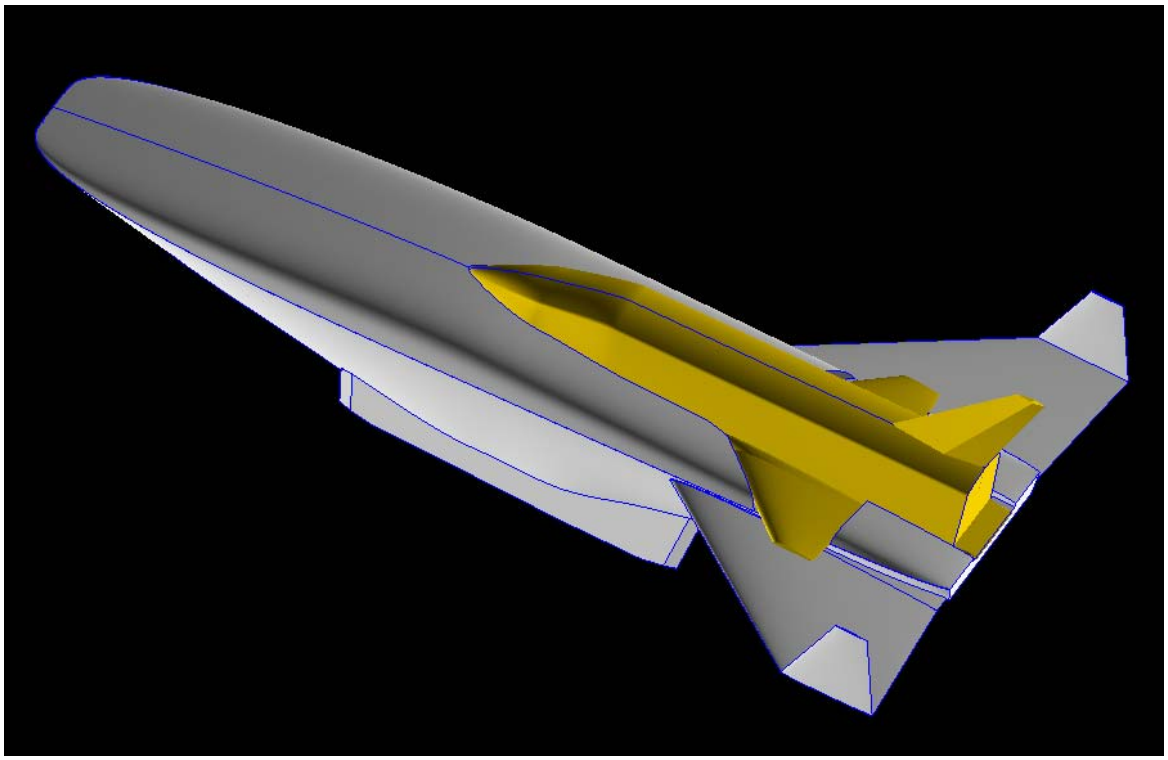


Figure 1. Two stage to orbit reference configuration.

Integration of Propulsion Data

Turbine Data

Propulsion engineers at NASA Langley and NASA Glenn agreed that for turbine performance data, the initial capability desired for IDEA was to develop an interface to a set of dynamic spreadsheets. These spreadsheets contain propulsion data for different turbine engines running different fuels, all generated offline with the Numerical Propulsion System Simulation (NPSS) code, but which include the ability to perform corrected airflow adjustments to allow for varying inlet and nozzle performance (if available). These spreadsheets have been integrated within the IDEA environment allowing the user to set input parameters via a graphical user interface and extract output data that will be used by the packaging (configuration) and trajectory disciplines. Figure 2 illustrates the data extracted from the turbine engine spread sheets.

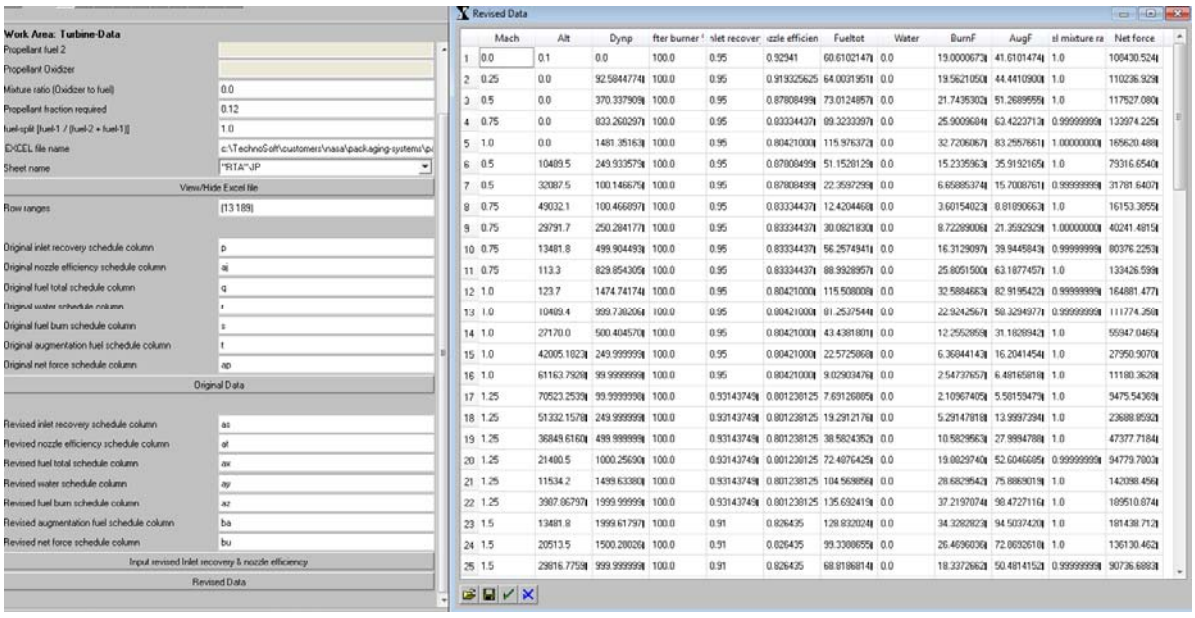


Figure 2. Example of turbine engine data extracted from data sheets.

Scramjet Data

Scramjet data can be generated either by running SRGULL through the SRGULL interface classes within IDEA, or by obtaining response surface equations that can be imported directly into IDEA as long as they conform to an agreed upon format. The SRGULL interface classes allow the user to generate an SRGULL deck given a keel line and cowl geometries. The generated deck can then be used to run SRGULL, and the output data can be extracted from the run. The main issues with this approach are that the data generated is only applicable to a single on-design flight condition and that automated execution of IDEA would require SRGULL to run successfully a majority of the time without user intervention, which is extremely challenging at certain flight conditions.

The second approach requires the propulsion engineers to perform Design Of Experiments (DOE) study running tens or maybe hundreds of SRGULL cases offline and from that generate a set of response equations that can be used for the different speed regimes (ram vs. scram), and fueling schemes. Once the response equations are generated and the associated keel line model is loaded, IDEA can then use this data to build a 3D vehicle and provide the propulsion performance data to trajectory discipline to evaluate the vehicle's integrated performance. Figure 3 illustrates the directory structure and data needed to build the SRGULL response data model branch. Different response equations are used based on the flight conditions specified by trajectory. For keel line designs that employ variable geometry, this setup allows the user to schedule the geometry as desired or to input a range of allowable values that can be used by trajectory as an independent variable in optimizing performance in flight.

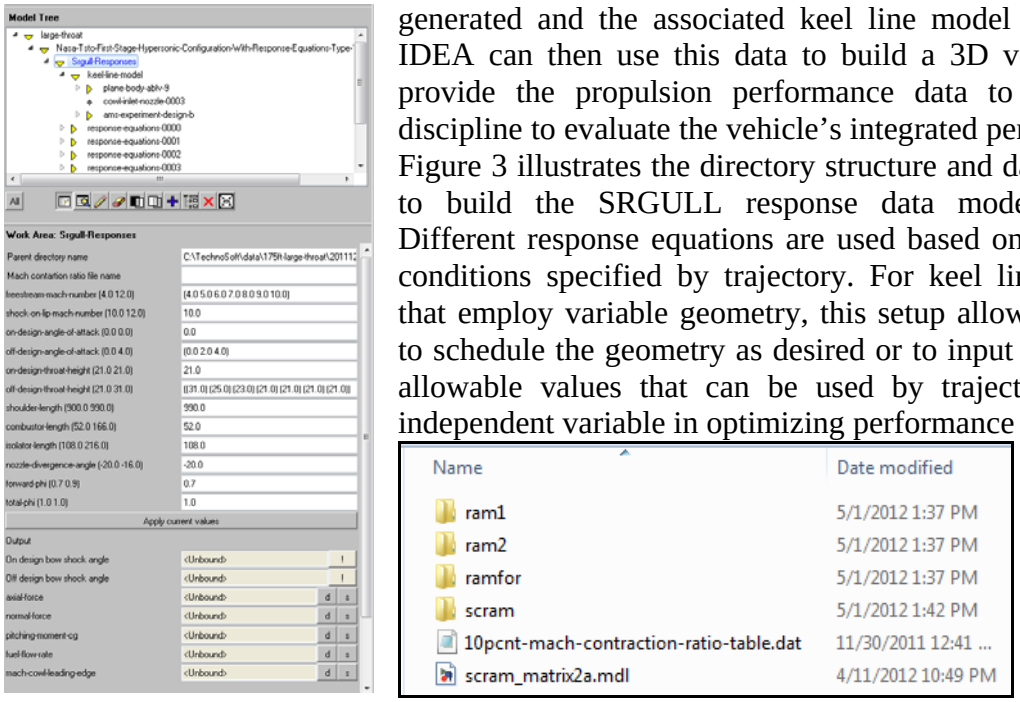


Figure 3. Model tree (left) and directory structure (right) used to generate the SRGULL response data model.

Rocket Data

For liquid rocket engines, a number of codes and spreadsheets have been integrated into the IDEA environment allowing the user to generate the data needed by the configuration and trajectory disciplines. The user can choose from an existing database of over 40 engines, ranging from the Saturn F-1 down to attitude control jets. The user can also create new "rubber" engines, with choices of over thirty-five fuels and half a dozen oxidizers, along with full control over the engine cycle and other design parameters. A screenshot of the liquid rocket engine design interface is shown in Figure 4.

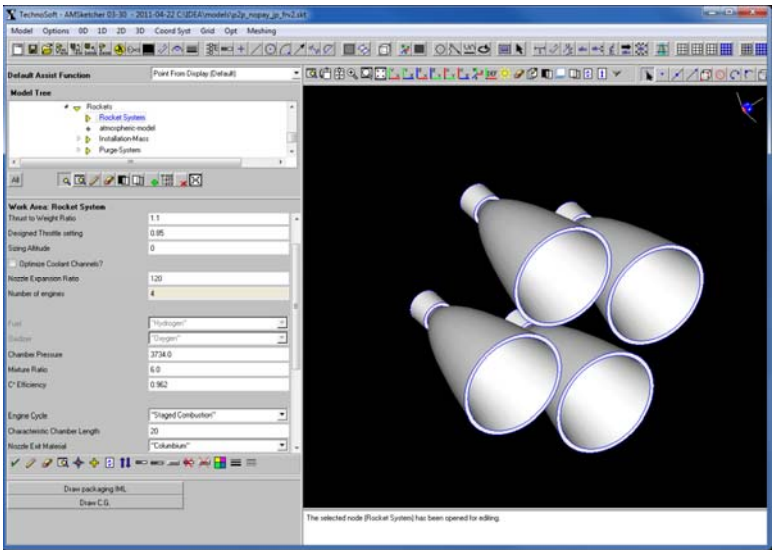


Figure 4. Screenshot of liquid rocket engine design interface.

Turbine Flowpath Geometric Representation

A set of classes has been developed and implemented to represent the lowspeed flowpath for the purposes of internal packaging. The overall geometric dimensions of the flowpath are extracted from the turbine engine data sheets described previously. The user has control over assigning different unit weights to the inlet and nozzle sections for mass properties purposes. The weight of the turbine itself is imported from the performance data in the spreadsheet. A set of lowspeed flowpath modules stacked side by side can be instantiated by the user, or the user can choose to model each flowpath separately. The propulsion data for the engine modules is represented as a single engine data set for use by the trajectory object, and IDEA handles any required scaling of the performance data given scale factors set either by the user or during a closure process (if the turbines are allowed to scale during closure). Figure 5 represents the lowspeed flowpath integrated into the vehicle, and constructed by linking the lowspeed flowpath geometry instance to a turbine engine data spreadsheet. The inset image in Figure 5 shows the lowspeed internal inlet and nozzle trimmed to the vehicle OML. Figure 6 shows an enlarged view of a single untrimmed lowspeed flowpath.

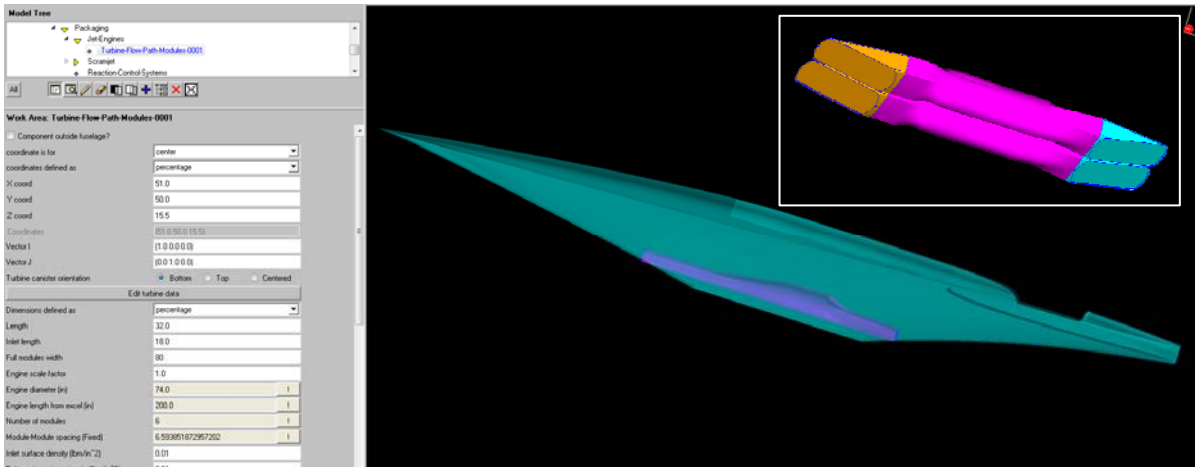


Figure 5. Lowspeed flowpath design interface. Image shows flowpath installation in airbreathing stage, with inset image showing inlet and nozzle trimmed to OML.

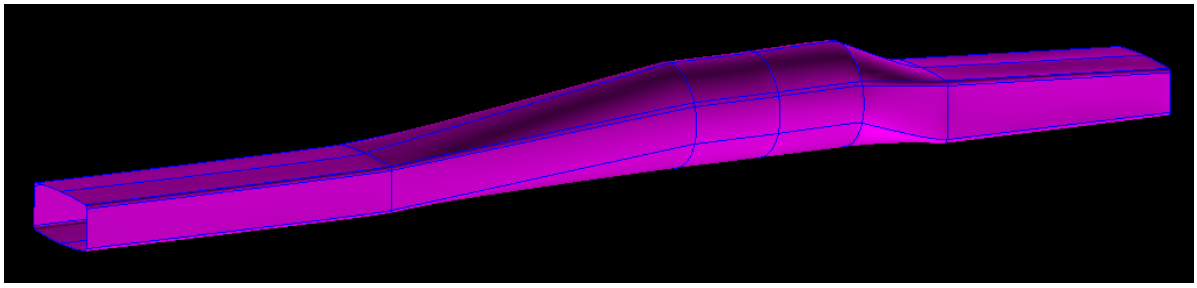


Figure 6. A zoomed in view of an untrimmed lowspeed flowpath.

Closure Methodology

A set of closure algorithms for closing rocket and airbreathing vehicle configurations was developed and implemented. Closure describes the process of modifying and/or resizing a vehicle to meet a specific set of mission requirements. In order to perform automated closure, methodologies had to be developed in several areas, including geometric scaling, automated internal packaging and performance analysis. Each of these is described below. Once these methods were in place, the analysis procedure (i.e. order of analyses to be performed) was implemented. Lastly, simple Newton-Raphson type iterative methods were then implemented to guide vehicle scaling in order to achieve closure.

Vehicle Scaling

In IDEA, two instances of the vehicle outer mold line geometry exist. The first, "as-drawn" geometry, is the version of the geometry that the user modifies to establish the initial vehicle shape and dimensions. The second, "scaling" geometry, is a rubberized version of the as-drawn and is the version modified by the closure routine. Each dimensional parameter used to establish the vehicle geometry (wing span, fuselage length, etc.) is associated with an X, Y or Z directional scaling factor. Additionally, the user can specify for each parameter whether it should be allowed to scale or not during sizing, as well as to set a minimum and/or maximum allowable value during scaling. This is useful, for instance, in the case of the upper stage where the payload is a fixed dimension size, but the vehicle is required to scale down in order to close. The user can set a minimum fuselage width and fuselage height constraint to maintain proper clearance for the payload. As the vehicle scales down (initially photographically, with X, Y, and Z scale factors all the same), it may reach one or more of those constraints. In such a case, the appropriate scale factor will stop decreasing, while others will continue until closure is achieved. Thus, it is important that all analyses be executed on each closure iteration, as non-photographic scaling is possible with this type of closure approach.

Tank sizing

A tank sizing option was added to the packaging system to allow for the automatic sizing of propellant tanks. This capability facilitates the automated re-packaging of tanks within the OML given updated propellant usage information, usually from trajectory analysis. It is the responsibility of the user to add the appropriate tank sizing relationships and instances. A tank's propellant fraction required (PFR) is computed based on the propellant usage of engines associated with the tank. This allows multiple engines to draw from, and in turn affect the sizing of a single propellant tank. There are two tank sizing classes:

1. *dm-packaging-tank-sizing-based-on-pfr-pfa-type-1-class*: An instance of this class requires the user to pick a forward tank and an aft tank, useful for instance in the sizing of main propellant tanks (fuel and oxidizer). The system will size the forward (fwd) and aft tanks until the ratio of available propellants in the tanks (fwd-pfa/aft-pfa) is equal to the ratio of required propellants (fwd-pfr/aft-pfr) within the specified tolerance.
2. *dm-packaging-tank-sizing-based-on-pfr-pfa-type-2-class*: This class allows the user to select a primary tank and a secondary tank. For this class, the secondary tank is sized until the ratio of available propellants in the primary tank relative to

the secondary tank (primary-pfa/secondary-pfa) is equal to the ratio of required propellants in the primary and secondary tanks (primary-pfr/secondary-pfr) within the specified tolerance. This class is useful in sizing pressurant tanks, as well as in cases of more than one primary fuel type (as may occur on a dual fuel airbreathing concept).

The ability to group tanks containing common propellants has also been added to the system. This feature allows the user to package multiple tanks into a vehicle (which may be advantageous given a certain shaped vehicle or particular structural layout) to accommodate all of the necessary fuel, but to treat those tanks with common propellants as a single tank tied to a particular engine in POST2. This feature maintains the packaging system compatibility with the POST2 "vehicle component weight model" feature.

Trajectory Analysis Automation

Several advanced features have been added to the trajectory interface to facilitate the automated closure process. One such feature is the ability to dynamically link one trajectory module to another, allowing trajectory branching. This feature is extremely useful in modeling the TSTO problem. A deck modeling the orbiter ascent from stage separation to orbit can be dynamically connected to the mated ascent simulation. Here, the orbiter simulation will take as its initial state the exact separation conditions achieved by the mated system. With this dynamic link, changes to the mated trajectory phase will automatically be propagated to other flight phases, and vice versa. This capability will ultimately allow trade studies and optimization to be performed on the entire TSTO concept, with impacts from all disciplines and flight phases included, in a more automated fashion. Trade study and optimization results should also be more accurate when performed with this capability, as a complete and exact set of state information will be transferred from flight phase to flight phase. With so much information being passed and multiple models to update, it is quite easy for errors to creep into the process when these types of analyses are performed manually.

An automated "crash" recovery feature has been added that allows the user to guide POST2, during automated execution, regarding what changes to make to the model to recover from an initial non-feasible starting solution. This type of situation occurs often when a large perturbation is applied to the vehicle (e.g. significant increase in mass), and trajectory optimization is attempted with a starting solution based on the previous, non-perturbed vehicle. For example, the orbiter ascent trajectory is typically guided by a table of pitch angle versus velocity. If vehicle mass is increased substantially, this profile will not provide sufficient lift and upward thrust vector to allow the vehicle to achieve orbit. It will typically crash back to Earth, and the run will terminate. The recovery feature allows the user to link a trajectory constraint to a trajectory input parameter. For the orbiter example, the user could create a constraint that the flight path angle at engine cutoff has to be positive. When the vehicle crashes, that constraint will not be met. The recovery feature would allow the user to connect that constraint with the pitch angle profile. When the case crashes, IDEA would identify that the linked constraint was not satisfied and increment the pitch profile by a user-specified amount. This adjustment will eventually

raise the flight profile enough that orbit can be attained, yielding an initial feasible solution for optimization to begin.

Additionally, an advanced run feature has been added that will execute POST2 in targeting mode prior to running optimization. This mode will allow POST2 to find a trajectory solution that satisfies all of the constraints prior and thus provides a feasible starting solution for optimization. This method has been found to aid optimization in achieving a solution more quickly.

Rocket vehicle closure

Currently, the closure process for the rocket-based vehicle class involves photographically scaling the vehicle, subject to scaling constraints described earlier. In the future, the closure process will likely include modification of certain design parameters based on the results of the aero, structural, and trajectory analysis codes to not only meet mission performance requirements but to also meet secondary performance objectives (e.g. take-off or landing speed). A data flow diagram of the closure process for the rocket-based vehicle class is shown in Figure 7.

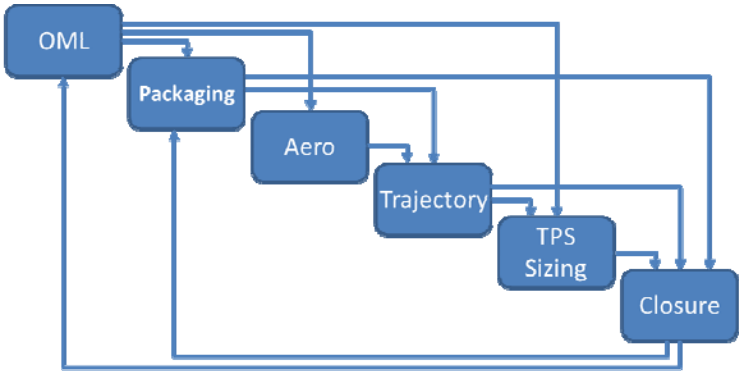


Figure 7. Data flow diagram for rocket-based vehicle closure.

The following is an outline of the closure process used to close the rocket-based vehicle with a single primary fuel and a single primary oxidizer:

1. An advanced trajectory run is executed. This involves turning off the optimization, running the trajectory in targeting mode only until all of the trajectory constraints are met, and then running the trajectory code with optimization on given the feasible starting solution. If all constraints are met after optimization is complete, then the closure loop continues. Otherwise, the vehicle closure has failed and the loop is terminated.
2. If structural analysis or TPS analysis are to be run as part of the closure process, then they are executed at this point. (Please note that structural analysis was not fully functional at this point, so mass estimating relationships were used to estimate the weight of the structures).
3. Propellant Fraction Required (PFR) is extracted from the trajectory object.
4. Propellant Fraction Available (PFA) is computed from the configuration instance, given updated TPS and structural weights computed from Step 2.
5. If PFR and PFA are roughly equal then there is no need to update the configuration's PFR (each engine within the packaging branch maintains its own

PFR, which are collected and used to size propellant tanks); else the configuration's PFR is updated using the one obtained from the trajectory object.

6. If the configuration's PFR is updated then a new PFA is computed.
7. The vehicle is rescaled based on the PFA and PFR.
8. If the new vehicle scale factor is roughly equal to the previous vehicle scale factor then the closure loop is done.
9. If the new vehicle scale factor is not roughly equal to the previous vehicle scale factor then trajectory is rerun, and a new PFR is extracted from the trajectory run. If all constraints are met then proceed to next step, else the closure loop terminates and generates a message that it failed to close and stop.
10. The closure loop stops if the maximum number of iterations has been reached, or when PFA from configuration instance is roughly equal to PFR from trajectory object. If the conditions above are not met then loop back to step (1).
11. Once the closure loop is done, the system checks whether the vehicle closed or not by making sure that all the constraints specified in the trajectory object have been met and that PFA from configuration instance is roughly equal to PFR from trajectory object.

Figure 8 illustrates the closure form associated with the rocket-based configuration. As seen in the figure, the user can control the number of allowable closure iterations, the PFA to PFR closure tolerance, as well as the maximum number of scaling iterations per closure iteration. There are two check boxes that allow the user to control whether or not TPS sizing and structural analysis should be executed during the closure run. Additionally, the user can control the tolerance on TPS and structural unit weights (results of analyses can be converted back to combined unit weights; the user can control how tightly they would like unit weights to be tracked). Finally, the user can set the PFA-PFR tolerance that will trigger TPS and structural analysis. This is useful when, for instance, the initial vehicle scale (PFA) is vastly different than the mission requirements (PFR), allowing the vehicle to scale one or more iterations prior to executing the more computationally expensive TPS sizing and structural analysis methods.

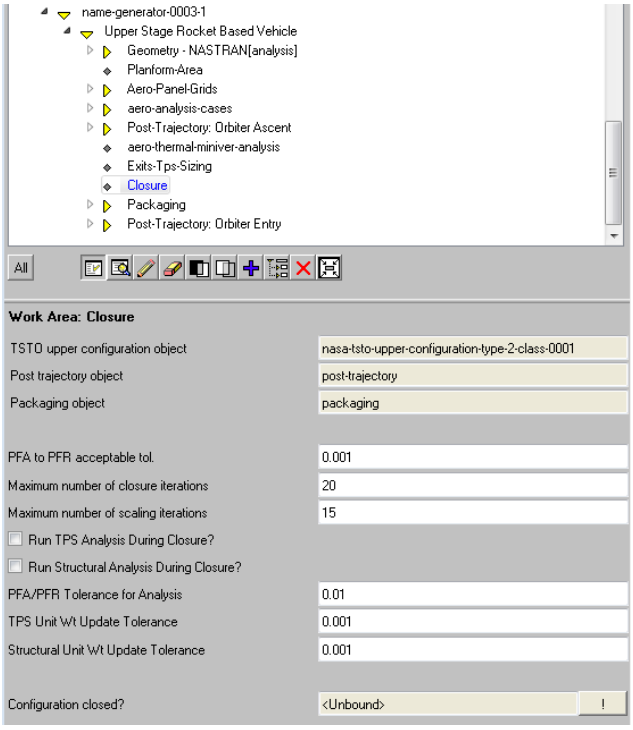


Figure 8. Model tree and the closure form associated with the rocket based configuration.

Airbreathing vehicle closure

As seen in Figure 9, the airbreathing booster that served as the model for the development of the vehicle class in IDEA contained a series of semi-conformal "box" tanks that were laid in between primary structural elements. In order to enable the closure process for the hypersonic airbreathing configuration, a set of classes that allow the user to automatically create sets of boxed tanks between specified bulkhead ranges was developed. As the substructures move, the tanks move and resize. In addition, any propellant tank in the system is able to identify the substructures associated with it, so as the tank is moved and resized as a result of the tank resizing algorithm, so are the substructures associated with it.

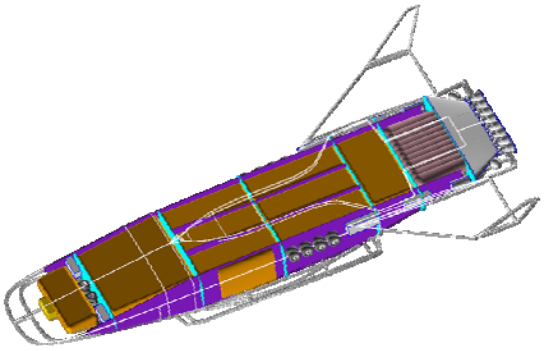


Figure 9. View of internal packaging of reference airbreathing vehicle.

The closure of the airbreathing vehicle configuration is complicated by the existence of multiple engines and operating modes, multiple fuel types and its unique internal arrangement. The steps in the closure process, however, are similar to those described in the previously for the rocket based vehicle. The one exception is that a tank sizing algorithm is executed after each time the vehicle is resized. This is done to guarantee that the various fuels and oxidizers in the vehicle meet the PFR extracted from the trajectory code. The airbreathing configuration contains the tank sizing branches described previously, where the user specifies the relationship between the primary and secondary tanks.

Closure Execution

A number of closure runs have been executed for the TSTO system shown in Figure 1. Closure runs were made with a 20,000 lbs and a 10,000 lbs upper stage payload mass. The hypersonic airbreathing closure class can "reference" or "point to" a rocket based vehicle closure instance, meaning that the upper stage will close first and act as a payload to the first stage. It takes roughly six hours to close (with 20,000 lbs. payload) and reclose the configurations (with 10,000 lbs.) running aero, propulsion, TPS sizing, and trajectory analysis on a single CPU PC running the Intel's Xeon processor. During each closure attempt, a number of closure / sizing iterations occur. After each resizing of the vehicle, all analyses are re-executed, including complete regeneration of the vehicle OML and internal packaging, and vehicle performance is recomputed. This guarantees that when closure is complete, all analyses have been executed on the "as-closed" vehicle. The closure criteria used in this case was that the difference in propellant fraction available and propellant fraction required after all analyses were executed was less than 0.1%. Each stage typically takes two to four iterations to close. A single "analysis" pass of the entire TSTO system takes about 50 minutes.

Conclusion

IDEA is a multi-disciplinary, multi-fidelity, geometry-centric environment that allows the user to design, configure, analyze, and close rocket-based and/or airbreathing-based vehicles in a timely manner using engineering and physics-based analysis codes. The under laying framework (AML), Common Computational Model (CCM), dependency relation network, and geometric reasoning features are key enabling technologies. An initial design capability has been developed around a two stage to orbit launch vehicle concept employing an airbreathing first stage and rocket second stage. A number of aerodynamic analysis codes have been integrated into the system. Data needed by the codes (including meshes, planform areas, etc.) are generated automatically. Propulsion data for a configuration is generated based on the type of engines and fuels associated with the configuration. Premin and Lanmin are used to generate the aerothermal environment for use by the TPS sizing codes. EXITS is used at this stage as the TPS sizing code. The data from TPS sizing code is used by the configuration discipline to compute mass properties. Structural and subsystem masses are computed using mass estimating relationships. The different disciplines are linked through the trajectory discipline where the data from disciplines come together and are used to generatively build the deck(s) needed to run the trajectory code. A fully automated system closure was executed on a PC laptop resulting in a “time to close” on a single mission of roughly three hours. A single “analysis” pass of the entire system (with no sizing) required just under one hour.