12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSM
17 - 19 September 2012, Indianapolis, Indiana

AIAA 2012-5520

Efficient Supersonic Air Vehicle Analysis and Optimization Implementation using SORCER

Scott A. Burton[1]
American Optimization LLC, Liberty Township, Ohio, 45044

Edward J. Alyanak[2] and Raymond M. Kolonay[3]
Air Force Research Laboratory, Wright-Patterson Air Force Base, Ohio, 45433

The Air Force Research Lab's Multidisciplinary Science and Technology Center is currently investigating conceptual design processes and computing frameworks that could significantly impact the design of the next generation efficient supersonic air vehicle (ESAV). To make the technological advancements required of a new ESAV, the conceptual design process must accommodate both low- and high-fidelity multidisciplinary engineering analyses. These analyses may be coupled and computationally expensive, which poses a challenge since a large number of configurations must be analyzed. In light of these observations, a design process described herein uses the SORCER (Service-Oriented Computing Environment) software to combine propulsion, structures, aerodynamics, performance, and aeroelasticity in a multidisciplinary analysis (MDA) of an ESAV. The SORCER engineering software provides the MDA automation and tight integration to grid computing resources necessary to achieve the volume of analyses required for conceptual design. Details of the SORCER implementation are illustrated through ESAV design studies using a gradient-based optimization method. A discussion of preliminary optimization results and SORCER grid computing integration is provided.

Nomenclature

AFRL	=	Air Force Research Lab
CFD	=	computational fluid dynamics
ESAV	=	efficient supersonic air vehicle
FEM	=	finite element model
LP	=	linear programming
MSTC	=	Multidisciplinary Science and Technology Center
IP	=	Internet Protocol
SORCER	=	Service-Oriented Computing Environment
SLP	=	successive linear programming
$ncon$	=	number of constraint functions
ndv	=	number of design variables
$f(\cdot)$	=	objective function
$\boldsymbol{g}(\cdot)$	=	constraint functions (*ncon*-by-1)
$\boldsymbol{x}$	=	design variable vector (*ndv*-by-1)

[1] Manager, Ph.D., AIAA Senior Member. E-mail: ScottABurton@AmOpti.com
[2] Project Engineer, Ph.D., AFRL/RQSE, AIAA Senior Member, E-mail: Edward.Alyanak@wpafb.af.mil
[3] Principal Engineer, Ph.D., AFRL/RQSE, AIAA Associate Fellow, E-mail: Raymond.Kolonay@wpafb.af.mil

I. Introduction

THE process of designing supersonic aircraft has well known engineering and financial challenges. The ability of a new aircraft design to meet various requirements is largely dictated by decisions made during the conceptual design phase. This is especially true for acquisition and life cycle costs. With this in mind, it is advantageous to include the best physics-based analyses possible given the schedule and resources. This typically results in the use of simple closed-form equations and statistical models during conceptual design. These computationally inexpensive tools facilitate the assessment of the thousands of designs required, but may lack the fidelity to uncover significant engineering risks in a design.

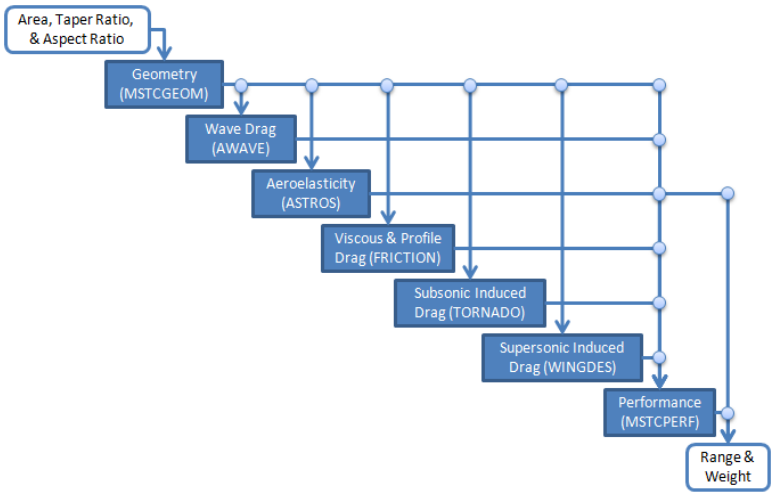

Figure 1. The ESAV MDA N^2 diagram includes geometry generation, aerodynamic analysis, aeroelastic analysis, and performance analysis.

In recent years, advances in computer processing speed and the prevalence of grid computing resources in industry have opened the door to more computationally expensive analyses during conceptual design. The conceptual design process described herein is based on the availability of such resources and incorporates analyses from several engineering disciplines. The analyses used in the efficient supersonic air vehicle (ESAV) multidisciplinary analysis (MDA) include techniques from traditional conceptual and preliminary design processes.

Details of the ESAV MDA are discussed in section *II. Multidisciplinary Analysis*. While the existence of grid computing facilities is a necessary condition for performing large-scale conceptual design studies, it is not by itself sufficient. Software to integrate the different engineering analysis codes and coordinate the execution of thousands of analyses remains an area of research. The study herein uses the SORCER (Service-Oriented Computing Environment) software to perform these functions.[1] Salient features of the SORCER ESAV application are discussed in section *III. Multidisciplinary Analysis Implementation*. The ESAV MDA is exercised in two optimization problems described in *IV. Design Studies*. Lastly, results of the ESAV design studies and conclusions are presented in sections *V. Results* and *VI. Conclusions*, respectively.

II. Multidisciplinary Analysis

The MDA used herein is a blend of conceptual and preliminary design methods from propulsion, structures, aerodynamics, performance, and aeroelasticity disciplines. The analysis process and data flow is shown in the ESAV N^2 diagram in Fig. 1. The process begins by parametrically generating discretized geometry suitable for several different analyses at varying fidelities. The geometry is used as input to compute several figures of merit of the aircraft, which include the aircraft drag polars, design mass, range, and aeroelastic performance. The different responses are evaluated for several flight conditions and maneuvers. These responses are then used to construct the objective and constraints of the multidisciplinary optimization (MDO) problem.

A. Airframe Geometry

The purpose of the airframe geometry application MSTCGEOM is to create geometry data for the physics-based analyses in the MDA. MSTCGEOM is compiled Matlab™ code and is considered medium fidelity. The application does not replace or replicate the extensive functionality of a commercial CAD package (i.e., a high fidelity geometry application); however, it does go beyond the functionality of a typical conceptual design tool. MSTCGEOM parametrically creates mid-plane panel aerodynamic models, and FEM stiffness and mass distribution models of the aircraft. The geometric features of the aircraft modeled include the fuselage and all of the lifting aircraft components (wings, vertical stabilizer, and horizontal stabilizers). The physics-based analyses that use the MSTCGEOM output include ASTROS, FRICTION, TORNADO, AWAVE, WINGDES, and MSTCPERF.[2-4]

MSTCGEOM is computationally inexpensive to execute and produces the required fidelity for the preliminary ESAV analyses described herein. In this manner, MSTCGEOM is a bridge between conceptual and preliminary design fidelities. MSTCGEOM may be used to make both parametric and topological changes to the airframe wing during the MDO. Figure 2 shows an example of the MSTCGEOM application output for the baseline ESAV.

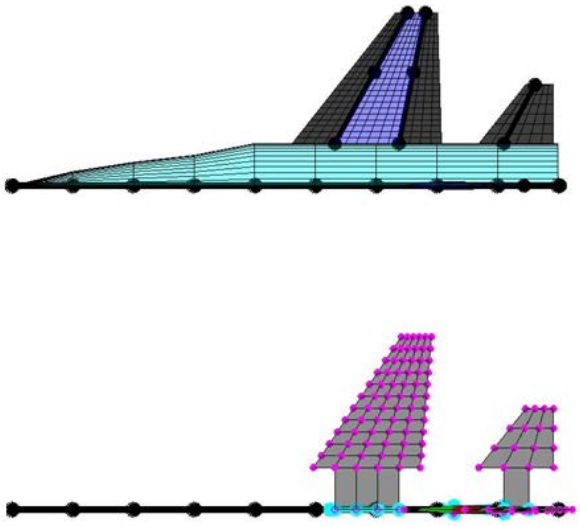

Figure 2. The MSTCGEOM application creates discrete geometry for ASTROS, FRICTION, TORNADO, AWAVE, WINGDES, and MSTCPERF applications (baseline ESAV half-span shown).

B. Aeroelasticity

Stress analyses for the ESAV are performed for four static aeroelastic maneuver cases at an altitude of 10,000 ft. Specifically, they are the following: 1) a 9g pull-up at Mach 0.9; 2) a 7.2g pull-up at Mach 1.2; 3) a negative 3g push-over at Mach 1.2; and 3) an anti-symmetric 100 deg/s aileron roll at Mach 1.2. ASTROS is used to do the aeroelastic analysis and structural sizing such that it optimizes the structural weight subject to stress constraints. (The structural sizing phase is independent of the ESAV MDO problem described later; the structural sizing sub-optimization is an integral part of the MDA and is performed for each configuration analyzed.) There are fifty-seven design variables in the structural sizing sub-optimization problem, which are distributed over the wing and tail surface spar, rib, and skin thicknesses. The ESAV is modeled with aluminum and 70 ksi is used as the maximum allowable stress.

C. Aerodynamics

To assess the aero performance of a configuration, estimations of zero lift drag and induced drag are required. The zero lift drag components estimated are viscous and profile drag. They are calculated using the FRICTION code.[3] Solutions for zero lift drag are computed at a number of altitude and Mach numbers.

The induced drag is estimated using the vortex lattice code TORNADO.[4] A TORNADO solution is computed over a range of angles of attack and Mach numbers to develop a set of drag polars. The combination of the results from FRICTION and TORNADO are used in the mission performance calculations for each configuration.

For the structural sizing problem described in the *Aeroelasticity* section above, the aerodynamic analysis used is within the ASTROS application. USSAERO is the aerodynamic solver used in the version of ASTROS applied for this paper. The USSAERO aerodynamic model contains all the lifting and control surfaces.

D. Propulsion

The aircraft engine performance is modeled using TERMAP (Turbine Engine Reverse Modeling Aid Program). TERMAP is an engine cycle performance program developed by Allison Engineering that uses a building block approach to predict both the steady state and transient performance of gas turbine engines.[5] The program is FORTRAN-based and requires the user to define engine component characteristics. TERMAP uses this information

to estimate flow path information at station numbers throughout the engine.

To improve the computational efficiency of the N^2 process in Fig. 1, TERMAP is used to build an approximation of thrust and specific fuel consumption over all flight conditions of interest. The approximation is used during the optimization to calculate range and optimal altitude trajectory for the flight leg of interest.

E. Mission Performance

The mission performance assessment involves calculating the aircraft range for a given Mach number. This is accomplished by using the design weight from ASTROS. This weight includes non-structural masses and internal fuel load, such that the design weight is the gross takeoff weight for the airplane. (Since a half-span model is used, the weight from ASTROS is half the takeoff gross weight.) The design weight along with the drag polars and the propulsion system performance assessment are used to calculate the range for the vehicle.

The range calculation assumes that the vehicle is flying at the optimal altitude for its initial weight. The altitude increases as fuel is burned, so the range reported is the distance traveled over the optimal altitude trajectory. For the ESAV study herein, the range calculation is done for a 6000 lb fuel burn at Mach 0.8.

III. Multidisciplinary Analysis Implementation

The practical large-scale MDO of an ESAV requires the MDA behind the objective and constraint functions be implemented such that the following hold true:

1) The evaluation of constraints and objectives is automatic and does not require user intervention.
2) The evaluation of constraints and objectives is reliable, repeatable, and accurate to the desired fidelity.
3) Constraint and objective evaluations that encounter errors or exceptions are communicated to the MDO algorithm gracefully and the MDO algorithm handles them appropriately.
4) The evaluation of objective and constraint functions for different designs is done in parallel using the computational resources available.

With these considerations in mind, the Java-based SORCER software is used to implement the ESAV MDA.[1] The MDA is then driven by an optimization algorithm implemented in Matlab via the SORCER *ModelClient* class.

A. SORCER Fundamentals

SORCER is a Java-based, network-centric computing platform that enables engineers to perform analyses and design studies in a very flexible, robust, and distributed computing environment. At the foundation of an engineering analysis within SORCER is the concept of a *service provider*. A service provider—or simply "provider"—is Java code implemented in accordance with SORCER standards that makes a number of Java methods available to remote users over a computer network.[6] These methods are referred to as the provider's *services*.

Providers typically provide services that leverage existing domain-specific analysis codes (e.g., CFD or FEM codes). These providers are referred to as *analysis providers*. The implementation of a given service in an analysis provider usually involves wrapping an engineering analysis executable with Java code. The executable is generally platform dependent and performs the bulk of the engineering-specific computations for a given service. An analysis provider's service may also enhance the functionality of the underlying executable with Java code or simply pass data to-and-from it.

Two common approaches to wrapping executables are to: 1) use the file system in conjunction with a system call from the analysis provider's service; or 2) use more sophisticated technology such as Java Native Interface to tightly couple the analysis provider's service to the executable. In the first case, the service writes an input file for the executable to read, invokes the executable via a system call, and concludes by reading the output file produced by the executable. The service may return the raw output files produced by the executable via a URL object or post-process the file and pass higher-level Java objects to the remote user. If errors or exceptions occur in an analysis provider's service, the caller is notified gracefully and may (or may not) take action to correct the problem. Once an analysis provider has been implemented to provide a number of services, it may be published from a computer on the network using SORCER. Remote users may then access the analysis provider's services via a small Java program called a *service requester*, or use simple SORCER command-line tools to access services directly.

For example, the service used in the ESAV study to create input files for ASTROS is called *doMstcGeomService*. This service is implemented by a provider called *Engineering Air Vehicle Provider*. The underlying executable for the *doMstcGeomService* is the compiled Matlab program MSTCGEOM. MSTCGEOM creates FEMs and other input decks required for the ESAV MDA. The argument for the Java method

doMstcGeomService is an object that contains all the information necessary to define the MSTCGEOM input deck, including the ESAV wing design variables (area, aspect ratio, and taper ratio). Using the input object, the Engineering Air Vehicle Provider's *doMstcGeomService* service writes an input deck for the MSTCGEOM Matlab executable, invokes MSTCGEOM via system call, and returns the URL of the ASTROS input file produced by MSTCGEOM to the remote user. In addition to the ASTROS input file, the MSTCGEOM executable also writes input files required by the *doTornadoService* and *executeFriction* services. Both of these services are also provided by the Engineering Air Vehicle Provider.

Like most services, the *doMstcGeomService* service returns multiple outputs that may be used by multiple downstream services to form a complete MDA. The returned objects from a service may be used directly or post-processed on the client-side (i.e., at the user's location) such that the output may be meaningful for design optimization. In the previous example, the output URL object pointing to the ASTROS input file is used directly by a downstream service. If a scalar-value was desired—e.g., to assign a value to an objective function during an optimization—the URL could be opened and parsed to yield a number. The means of reducing output from services in this manner is accomplished with the SORCER *Filter* subclasses, discussed later.

To provide a layer of abstraction from a specific implementation of a service (i.e., Java method), providers implement Java interfaces to identify the services they provide. (Java interfaces are defined by a list of method signatures, which are composed of a method name, argument types, and return type.) These Java interfaces are referred to as *service types.* In this manner, a remote-user is generally only concerned about finding a provider that implements the service type containing the service of interest. The specific provider usually does not matter, since it is the service that the user desires. Many providers on the network may have different implementations of the same service in which case they would all implement the same interface (i.e., service type). If a user desires to select a specific provider for a given service, the user may specify the name of the provider in addition to the service type.

To use SORCER providers that are published on a network, a user must specify the name of the service, the service type that contains the service of interest, and the arguments for the service. The user *does not* have to specify the hostname, port, or IP address of the computer the service is running on. The arguments for all SORCER services are instances of the *Context* class. A Context object—or "context"—is a generic container that encapsulates name-value pairs. In this manner, a context possesses its own namespace for identifying argument objects for a service. The service name and service type are encapsulated in the SORCER *ServiceSignature* class. (Instances of *ServiceSignature* are referred to as "signatures.")

The combination of the service name, service type, and a context are encapsulated in an instance of the SORCER *Task* class. A *Task* instance—or "task"—represents the basic unit of work that service providers operate on. Since tasks involve using remote services, a layer of abstraction exists to encapsulate both remote and locally executed units of work. The *Evaluator* class and its subclasses are the entities responsible for doing computational work via their *evaluate* method from both remote and local resources in SORCER.

An *Evaluator* instance—or "evaluator"—that encapsulates a task is called an *exertion* evaluator. The word "exertion" implies the *remote* use of service providers. This is in contrast to other subclasses of *Evaluator* that are capable of doing simple calculations locally when the heavy lifting nature of an engineering analysis service is not required. For example, the *ExpressionEvaluator* class enables a user to combine *Var* instances in algebraic equations without using a remote service (the *Var* class is discussed later). In this light, evaluators are the entities responsible for either defining the process of how data is produced via remote services or producing the data themselves locally via their own logic.

Var instances—or "vars"—are used to form both independent and dependent variables in SORCER. In the case of independent variables, vars have an instance of *IndependentEvaluator* that contains the value of the independent variable. The *IndependentEvaluator* object acts as a simple container to store a value and does not perform any calculations or use services on the network.

Dependent vars are typically created to implement scalar-valued mathematical functions, such as objective, constraint, or other mathematical functions for use in analysis or optimization studies. They may also be to form composite functions. Vars contain the objects necessary to produce—rather than store—the values they represent, which may include *Evaluator*, *Filter*, and *Persister* instances. The three types of objects are used in a specific sequence when the method *getValue* is invoked on a dependent var.

When the *getValue* method is called on a dependent var, the evaluator's *getValue* method is subsequently called. The evaluator's *getValue* method then, in turn, calls *getValue* on its argument vars to ensure their respective values are current before proceeding. (If a var represents a function dependent on other vars, i.e., argument vars, the evaluator of the var representing the function holds references to those argument vars.) The evaluator then checks to see if it is out of date due to argument vars changing since the last invocation of *evaluate*. If the argument vars have not changed, the evaluator returns its current value without further processing. In this manner, all calls to *evaluate* are demand-driven, i.e., the *evaluate* method is invoked only if the evaluator's arguments have changed since the last invocation of *evaluate*. As described earlier, the *evaluate* method may use a remote service implemented by a provider on the network to perform the necessary calculations, or the calculations may be done locally via its own logic. Once the evaluator's *evaluate* and *getValue* methods complete and return a new value to the var, an instance of a *Filter* subclass may be employed.

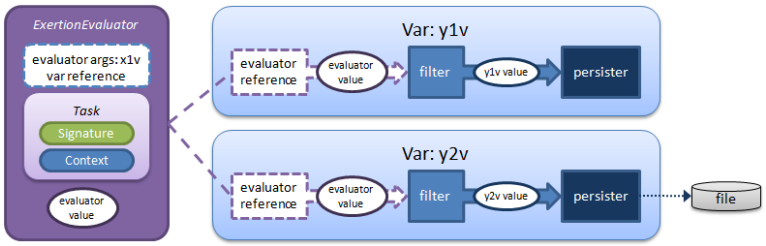

Figure 3. A single *Evaluator* object may be referenced by multiple *Var* objects employing different filters to reduce aggregate data to scalar values; the values may be then written to a file (or object field) using a var's persister.

Since exertion evaluators typically return aggregate forms of data, reducing the output from them is often necessary. This data reduction is accomplished using the *Filter* subclasses. Using a *Filter* subclass object—or "filter"—on the returned objects from an evaluator allows users to reduce or transform the aggregate data produced by evaluators to yield new objects. The result of filtering is often a scalar value of a mathematical function. In the case where an evaluator produces aggregate data, the evaluator may be referenced by a number of different vars with different filters to yield different values, see Fig. 3. A series of filters may be combined to form a *filter pipeline*, such that downstream filters operate on the output of upstream filters. The resulting value returning from the filter (or pipeline filter) is then passed to the var's *Persister* instance if it exists.

The *Persister* instance—or "persister"—uses the value from the filter as an argument to its *setValue()* method. Depending on the specific subclass of *Persister* used, the persist method may write the var value to a file or set a field on an object. Once the persister is done executing, the var value is returned to the caller of the var's *getValue* method.

An *OptimizationModel* instance—or "model"—is a collection of any number of vars that define a specific optimization problem. Figure 4 illustrates a simple example of an *OptimizationModel* called *model1*. The model contains three vars; one independent and two dependent. The var *y1v* is a function of *x1v*, which is the independent var. The var *y2v* is a function of *y1v*, thus it is a composite function.

The vars are the implementation of the design variables, optimization objective function, and constraint functions. Additional vars may be included for monitoring other figures-

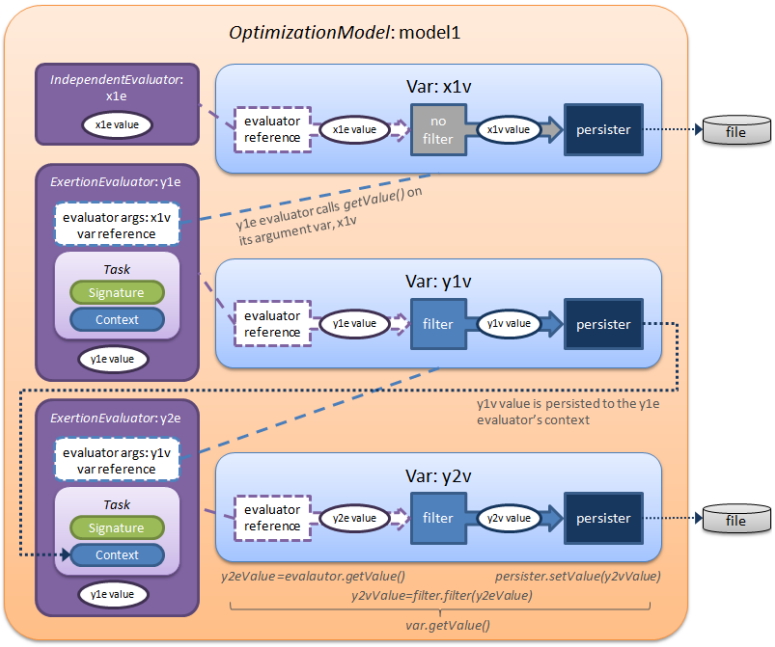

Figure 4. An *OptimizationModel* is a collection of *Var* objects; each var references an evaluator and may be referenced by other vars to form composite functions.

of-merit. Once a model is created, it may be published as a provider for users to access remotely. This type of provider is referred to as a *model provider* to distinguish it from analysis providers. The model provider behaves like shared memory that is accessible to remote users over the network. There is only one state of the model, which is subject to manipulation by remote users. There are two techniques for users to interact with a published model provider: 1) via a single model query; or 2) via a table model query.

A single model query is done from a requestor or SORCER command-line tool. In both cases, a query object is constructed containing the design variable var names and values and the var names of the objective and constraint functions that the user wishes to calculate. The query also contains the name of the model provider, so that it may be found on the network.

In contrast to analysis providers, accessing model providers over the network does not require users to specify a service type and service name. Since all *OptimizationModel* instances have only one service type with a single service, the information is implied. This does, however, necessitate that the user specify the model provider name when multiple models are published.

Once the query is constructed, it is executed by the user. The published model provider receives the query object and proceeds to invoke *setValue* on all the design variables. Once the *setValue* calls are completed, the model begins calling *getValue* on the user-specified objective and constraint functions. After all the *getValue* calls are complete, the query object is returned to the remote user with the updated var values.

The second technique of using models is via a table query. Rather than passing a single set of design var values to the model provider, a user may construct a table of runs containing the names of the design vars and their values for each run. The table is added to the query with the name of the model provider and the var names of the objective and constraint vars the user wishes to calculate. Since a table of runs is being sent to the model provider, the user may specify the number of runs the model provider is to execute in parallel.

Once the table query is constructed and sent to the model provider, the model provider creates new child instances of itself for parallel execution of the table. (The child instances are newly constructed objects, such that the states of their respective vars are not inherited from the model provider.) The model provider then creates a thread for each child model and begins to *setValue* and *getValue* on the vars as in the single model query. Once all the threads complete, the var values for each run are returned to the caller in a table object.

In contrast to the single model query, the table query does not affect the state of the parent model provider's vars. The child models are isolated from one another and the parent model, and are used exclusively to perform the user-requested calculations. Once the calculations are complete, the child models are discarded and the states of the parent model vars remain unaffected.

B. SORCER Space Computing

Large numbers of engineering analyses are generally required to perform conceptual-level aircraft design. This significant computational burden is addressed at the AFRL MSTC by using the SORCER software. The network-centric approach of SORCER enables the use of heterogeneous computing resources, including a variety of operating systems, hardware, and software. Specifically, ESAV studies performed at the AFRL MSTC use SORCER in conjunction with a mix of Linux-based cluster computers, desktop Linux-based PCs, Windows PCs, and Macintosh PCs. The ability of SORCER to leverage these resources is significant to MDO applications in two ways: 1) it supports platform-specific executables that may be required by an MDA; and 2) it enables a variety of computing resources to be used as one entity (including stand-alone PCs, computing clusters, and high-performance computing facilities). The main requirements for using a computational resource in SORCER are network connectivity and Java compatibility. SORCER also supports load balancing across computational resources via the JavaSpaces technology, making the evaluation of objective and constraint functions in parallel a simple and a dynamically scalable process. [7]

SORCER employs JavaSpaces technology to implement a loosely-coupled distributed computing system. JavaSpaces enables different processes on different computers to communicate asynchronously in a reliable manner.[7] Using this technology, SORCER implements a self-load balancing grid computing system that can dynamically grow and shrink during the course of an optimization study.

A JavaSpace—or "space"—is Java code running on a computer that is accessible on the network in the same manner as service providers, see Fig. 5. The space provides a type of shared memory where exertion evaluators can put tasks they wish to be processed by service providers. Service providers, in turn, use Jini discovery mechanisms upon startup to find spaces on the network and monitor them for tasks with their service type. If a service provider sees a task it can operate on in a space, and the task has a flag indicating it has not been processed, the provider takes the task from the space. The provider then executes the appropriate service and returns the task to the space with a flag indicating the task has been processed. Once the task has been returned to the space, the process that initially wrote the task to the space detects the returned task and checks to see if it has been processed. If the task indicates it has been processed, the evaluator removes the task from the space.

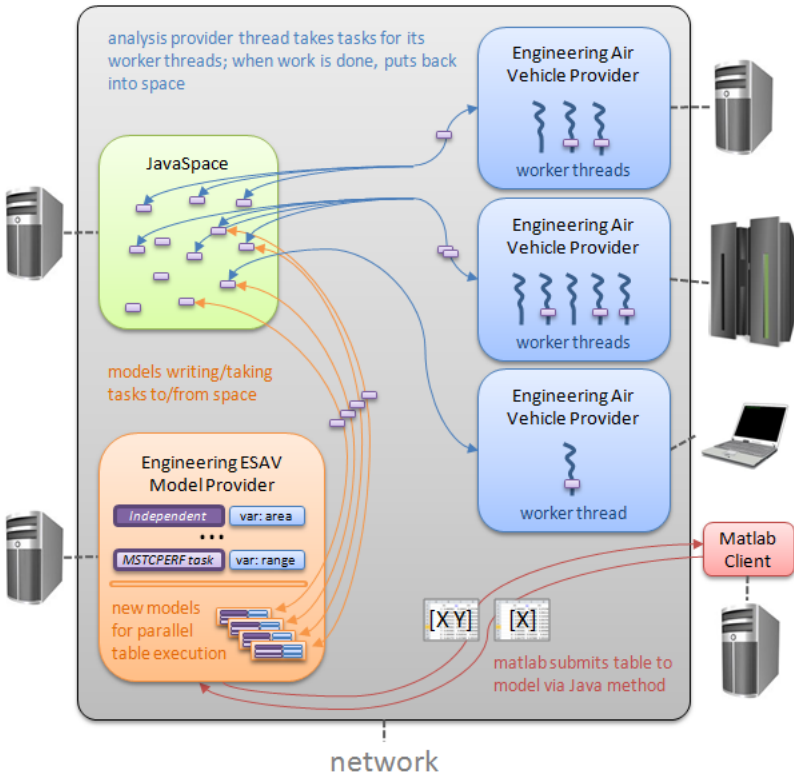

Figure 5. SORCER uses JavaSpaces technology to provide a flexible, dynamic grid computing facility for ESAV optimization studies.

To achieve the load balancing across multiple computers, a service provider may be configured to have a fixed number of worker threads. The number of worker threads determines the number of tasks a provider can process in parallel. By configuring the number of worker threads for a specific service provider on a specific computer, the provider can self-load balance the computer it is hosted on (assuming that it is the only service provider operating on the host computer).

IV. Design Studies

The design of a supersonic aircraft involves assessing many figures of merit and the tradeoffs between them. In this study, a preliminary exploration of the design space is performed through univariate parametric sweeps of the design variables. In this manner, each design variable is independently swept from its lower bound to its upper bound in equal steps. This process is costly, but it serves three important purposes: 1) the sweeps provide a set of results that engineers may easily use to assess the validity of the MDA implementation; 2) the results may be used to screen design variables and responses for significance, which may result in a smaller MDO problem; and 3) the sweep results are used to calculate the sensitivities for the first iteration of the gradient-based optimization process. Once the preliminary study is completed, the MDO problem is solved using a robust successive linear programming (SLP) technique.

A. Optimization Problem Statement

A single objective function is identified along with nonlinear and side constraints to exercise the ESAV model in SORCER. The optimization problem takes the general form:

$$Minimize\ f(\boldsymbol{x}) \tag{1}$$

$$subject\ to\ \boldsymbol{g}(\boldsymbol{x}) \leq \mathbf{0} \tag{2}$$

$$and\ \boldsymbol{x}^{lb} \leq \boldsymbol{x} \leq \boldsymbol{x}^{ub} \tag{3}$$

$$where\ \boldsymbol{g} \in \mathbb{R}^{\boldsymbol{ncon}}\ and\ \boldsymbol{x} \in \mathbb{R}^{\boldsymbol{ndv}}. \tag{4}$$

The objective is to minimize the weight of the ESAV, which includes the fuselage and lifting surface weights. In this manner, the life cycle cost of the aircraft is kept down due to its strong correlation with takeoff gross weight.[8]

Two optimization problems are solved with different range constraints. In the first case, the cruise leg range is constrained to be above 1500 mi at Mach 0.8 for 6000 lb of fuel. In the second case, the range is increased to 2500 mi. The design variables are wing area, taper ratio, and aspect ratio. The lower and upper bounds on the design variables are: 100 and 300 ft^2 for wing area, 0.6 and 5.0 for aspect ratio, and 0.1 and 1.0 for taper ratio, respectively.

B. Optimization Algorithm

The optimization technique applied is an adaptation of the Frank-Wolf implementation of the successive linear programming (SLP) method.[9] The SLP method used herein is tailored for parallel computing on a large number of CPU cores such that gradient and line search calculations are executed in parallel.[10] The algorithm is tolerant to failed runs in both the gradient and line search phases of an optimization iteration. This robustness comes with the computational cost associated from the extra analyses required over that of a typical first-order finite-difference technique.

The optimization begins by calculating the gradient of the objective and constraint functions at the initial design point. The gradients are then used to form a constrained linear programming (LP) problem where the optimum is found using an LP solver. In traditional implementations of gradient-based methods, a line search follows that involves sequentially searching for a minimum along a line from the initial design to the estimated optimum. This approach is inefficient given the computing resources available to most engineers today.

Since many CPU cores are available on a single computer, cluster, or high-performance computing facility, the line search is converted to a partial enumeration of the line. The number of designs evaluated on the line depends on the computational resources available, the anticipated nonlinearity of the search, the computational expense of the analysis, and the schedule of the MDO project. Once the line is partially enumerated, the local optimum may be found via a simple curve fit to a quadratic polynomial. Using the local optimum determined from the partial line enumeration, a subsequent iteration begins with a gradient evaluation at this point. The SLP process continues based on optimization convergence criteria or schedule considerations.

The SLP algorithm is implemented using Matlab and uses the *lingprog* LP solver to determine search directions. The responses required to calculate the objective and the constraints are supplied via the SORCER *Engineering ESAV Model Provider*. The communication between Matlab and the Engineering ESAV Model Provider is done by instantiating the SORCER *ModelClient* class within a Matlab session. The *ModelClient* class provides a simple interface for setting design variable values, running the MDA, and obtaining the responses necessary to form the objective and constraint functions.

V. Results

Figure 6 shows the planforms for the baseline (approximately 1550 mi range), the 1500 mi range design, and the 2500 mi range design. The optimization iteration history is shown in Fig. 7 for the design variables, constraint (range), and objective (weight). Figure 8 plots the results of the ASTROS structural sizing sub-optimization performed on the three ESAV designs. The figure shows the optimal thicknesses of the skins, ribs, and spars for the three configurations.

In the first case with the 1500 mi range constraint, the wing area was reduced to minimize weight. Since the baseline range was greater than 1500 mi, the optimizer had the design freedom to reduce the objective function in this manner. Taper ratio was also reduced having a similar effect. The structural sizing sub-optimization was driven in this case by two of the four maneuvers: the 9g pull-up at Mach 0.9; and the 7.2g pull-up at Mach 1.2.

The second optimization problem solved was for the 2500 mi range constraint. The planform of the wing in this case became long and slender with a high aspect ratio, high wing area, and low taper ratio. Since the initial design was infeasible, a considerable amount of weight was added via increases in wing area and aspect ratio to achieve the 2500 mi range. The structural sizing sub-optimization was driven by the 9g pull-up at Mach 0.9 maneuver.

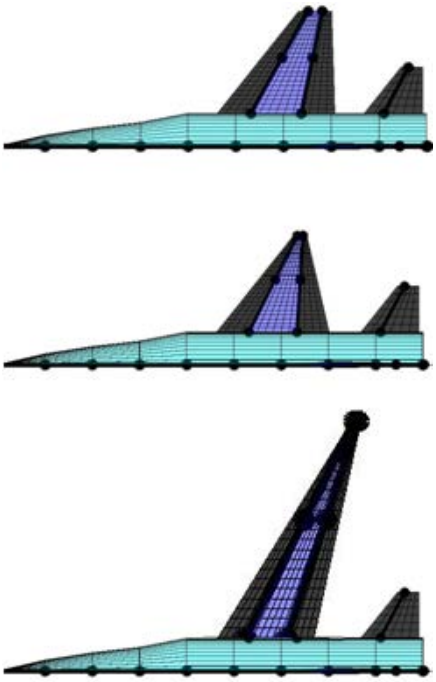

Figure 6. The ESAV optimization result half-span planforms: baseline 1550 mi range (top); 1500 mi range (middle); and 2500 mi range (bottom).

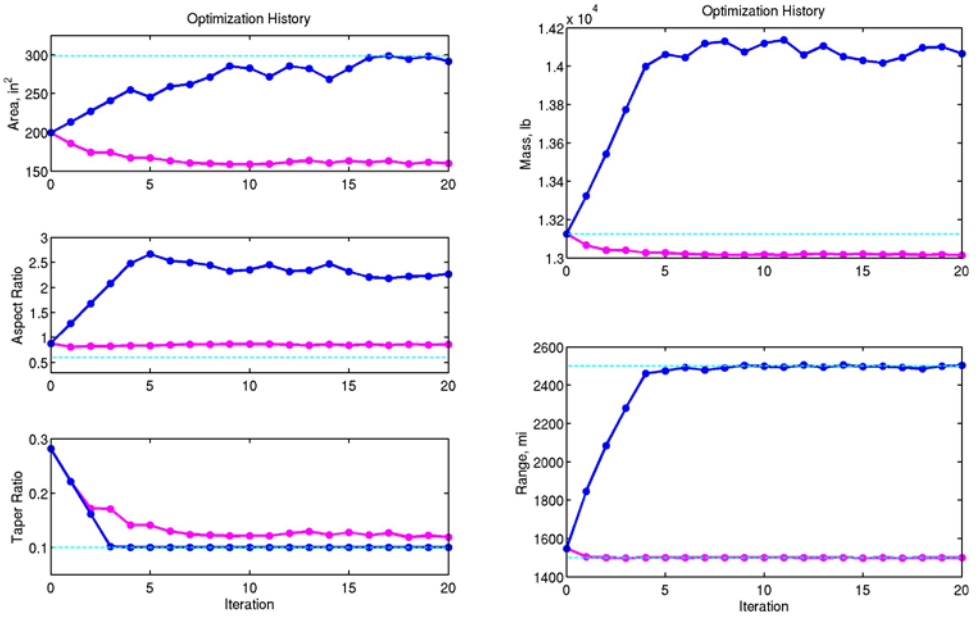

Figure 7. Results from ESAV optimization illustrate how the design variables, objective and constraint functions are manipulated by the SLP technique (1500 mi range constraint, magenta; 2500 mi range constraint, blue).

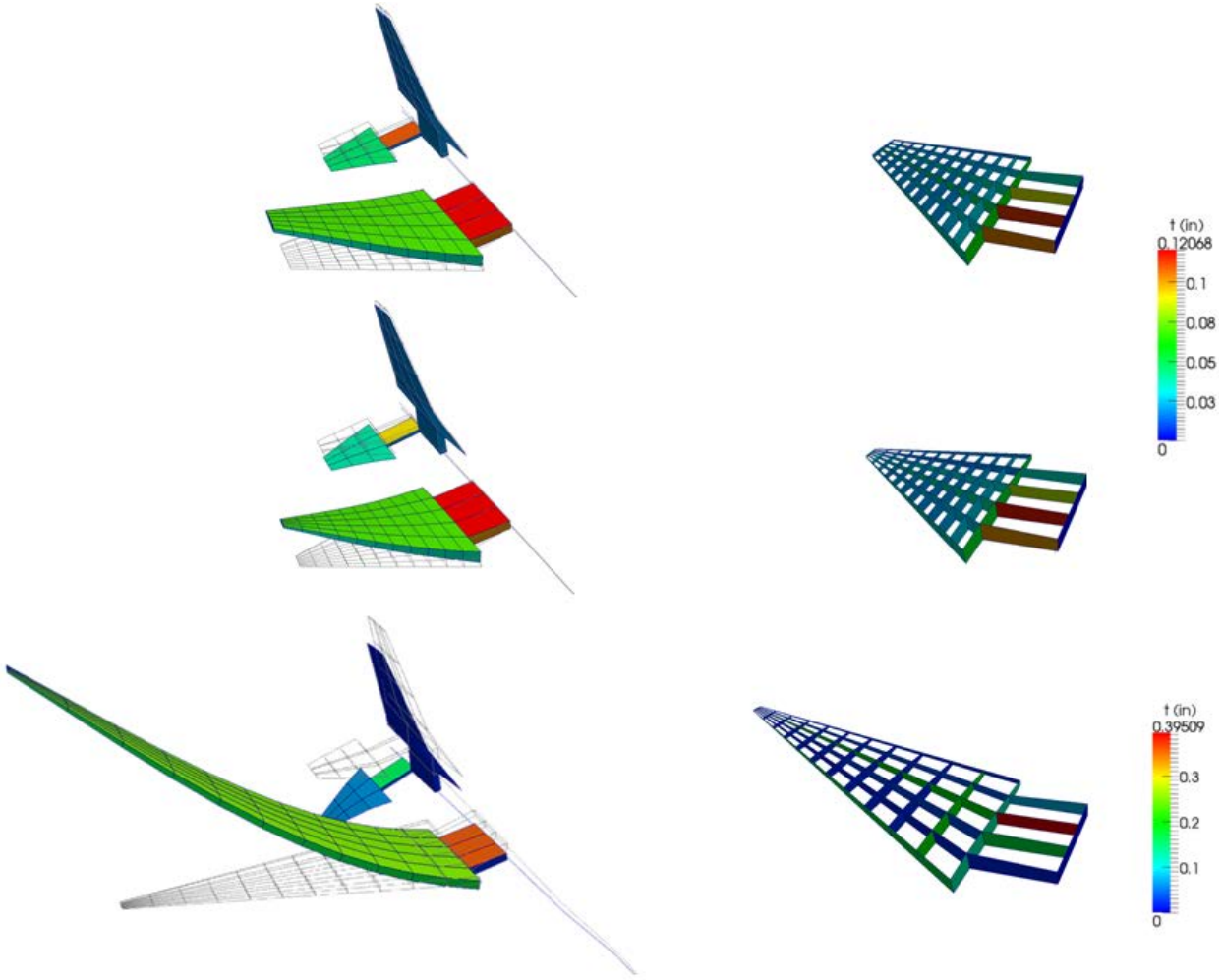

Figure 8. The ESAV planform and structural design thicknesses from the ASTROS structural sizing sub-optimization problem are shown above for the 1550 mi range baseline design (top row), the 1500 mi range design (middle row), and the 2500 mi range design (bottom row) optimal designs. (The left column shows the thicknesses for the wing skins and the right column shows the thicknesses of the ribs and spars.)

VI. Conclusions

The SORCER engineering software was used successfully in the design of two prototype ESAVs. The results from both optimization cases exhibited the correct trends consistent with historical aircraft design. These results provide a degree of validation of the implementation of the Matlab SLP code, the SORCER ESAV model, the SORCER providers, and the JavaSpaces technology.

The use of the JavaSpaces technology for parallel distributed computing proved reliable and efficient. It was a straightforward process to add computers to the SORCER grid as needed during the course of the two optimization studies. This flexibility proved valuable as the number of computers available varied from day-to-day.

Lastly, the application of a gradient-based solution technique capable of handling failed design points was demonstrated successfully. The robust nature of the SLP technique and its ability to perform gradient and line search functions in parallel made the ESAV optimization studies possible in a short period of time with minimal user intervention.

References

[1]Kolonay, R. M., & Sobolewski, M., "Service ORiented Computing EnviRonment (SORCER) for Large Scale, Distributed, Dynamic Fidelity Aeroelastic Analysis & Optimization," *International Forum on Aeroelasticity and Structural Dynamics.* Paris, 2011.

[2]Neill, D., & Herendeen, D., *ASTROS Enhancements: Volume I – Astros User's Manual*, Wright Laboratory WL-TR-96-3004, 1995

[3]Mason, W. H. (2002). *Software for Aerodynamics and Aircraft Design*. Retrieved 2012, from Skin Friction/Form Factor Drag estimation: http://www.dept.aoe.vt.edu/~mason/Mason_f/MRsoft.html#SkinFriction

[4]Melin, T. (2000). A Vortex Lattice MATLAB Implementation for Linear Aerodynamic Wing Applications. Royal Institute of Technology

[5]Macinnis, D. V., "A Simplified Cycle-Matching Transient Simulation of a Turbofan Engine," *AIAA Joint Propulsion Conference*, Indianapolis, 1994.

[6]Newmarch, J., *Foundations of Jini 2 Programming*, Apress, Inc., 2006, Chap. 1.

[7]Freeman, E., Hupfer, S., and Arnold, K., *JavaSpaces Principles, Patterns, and Practice*, Addison Wesley Longman, Inc., 1999, Chap. 1.

[8]Raymer, D., *Aircraft Design: A Conceptual Approach*, 2nd ed., AIAA Education Series, 1992, Chap. 18.

[9]Reklaitis, G. V., Ravindran, A. and Ragsdell, K. M., *Engineering Optimization: Methods and Applications*, John Wiley & Sons, Inc., New York, 1983, Chap. 8.

[10]Burton, S. A., Prakash, C., and Machnaim, J., "Multistage Low Pressure Turbine Airfoil Shape Optimization using the C^3 Process," *3rd AIAA Multidisciplinary Design Optimization Specialist Conference*, Honolulu, 2007.