Advances in Aircraft and Spacecraft Science, *Vol. 4, No. 3 (2017) 297-316*
DOI: http://dx.doi.org/10.12989/aas.2017.4.3.297

Service ORiented Computing EnviRonment (SORCER) for deterministic global and stochastic aircraft design optimization: part 1

Chaitra Raghunath[*1], Layne T. Watson[1,2,3a], Mohamed Jrad[3b], Rakesh K. Kapania[3c] and Raymond M. Kolonay[4d]

[1]*Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061, USA*
[2]*Department of Mathematics, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061, USA*
[3]*Department of Aerospace & Ocean Engineering, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061, USA*
[4]*AFRL/RQVC, 2210 8th Street, Bldg. 146, Wright-Patterson Air Force Base, Dayton, OH 45433, USA*

(Received May 18, 2016, Revised August 8, 2016, Accepted September 20, 2016)

Abstract. With rapid growth in the complexity of large scale engineering systems, the application of multidisciplinary analysis and design optimization (MDO) in the engineering design process has garnered much attention. MDO addresses the challenge of integrating several different disciplines into the design process. Primary challenges of MDO include computational expense and poor scalability. The introduction of a distributed, collaborative computational environment results in better utilization of available computational resources, reducing the time to solution, and enhancing scalability. SORCER, a Java-based network-centric computing platform, enables analyses and design studies in a distributed collaborative computing environment. Two different optimization algorithms widely used in multidisciplinary engineering design-VTDIRECT95 and QNSTOP-are implemented on a SORCER grid. VTDIRECT95, a Fortran 95 implementation of D. R. Jones' algorithm DIRECT, is a highly parallelizable derivative-free deterministic global optimization algorithm. QNSTOP is a parallel quasi-Newton algorithm for stochastic optimization problems. The purpose of integrating VTDIRECT95 and QNSTOP into the SORCER framework is to provide load balancing among computational resources, resulting in a dynamically scalable process. Further, the federated computing paradigm implemented by SORCER manages distributed services in real time, thereby significantly speeding up the design process. Part 1 covers SORCER and the algorithms, Part 2 presents results for aircraft panel design with curvilinear stiffeners.

Keywords: service-oriented computing; federated computing; deterministic global optimization; stochastic optimization; multidisciplinary design

∗Corresponding author, M.Sc., E-mail: chaitra@vt.edu
[a]Professor, E-mail: ltwatson@computer.org
[b]Post-doctoral Fellow, E-mail: jmohamed@vt.edu
[c]Mitchell Professor, E-mail: rkapania@vt.edu
[d]Ph.D., E-mail: raymond.kolonay@us.af.mil

http://www.techno-press.org/?journal=aas&subpage=7 ISSN: 2287-528X (Print), 2287-5271 (Online)

1. Introduction

This paper discusses the integration of two global optimization algorithms, VTDIRECT95 and QNSTOP, into a SORCER framework. SORCER is a large-scale, distributed computing environment for high fidelity multidisciplinary design optimization (MDO). The algorithms VTDIRECT95 and QNSTOP were chosen because of their relevance to aerospace engineering and their scalability on distributed computing applications. The integration of VTDIRECT95 and QNSTOP with SORCER is illustrated by design studies of panels having curvilinear blade-type stiffeners under multiple loading conditions. The mass of the panel is minimized subject to constraints on buckling, von Mises stress, and crippling criterion using the algorithms VTDIRECT95 and QNSTOP on a SORCER grid.

1.1 Multidisciplinary design optimization

Aerospace systems today exhibit strong interdisciplinary interactions and require a multidisciplinary, collaborative approach (Raymer 2006). Aircraft design, an inherently complex multidisciplinary process, comprises determining aircraft configuration variables satisfying all the design constraints for all the disciplines involved. Multidisciplinary design optimization aims to achieve an optimal design over all the disciplines integrated together. The first step in the design process, conceptual design, is characterized by an extensive exploration of the design space and analyses of a very large number of potential design configurations in order to assess the impact of design variables on the aircraft performance. Conceptual design of complex systems requires optimization with a large number of design variables belonging to multiple disciplines.

Traditional conceptual design focuses on low fidelity models, and is carried out in the initial design phases when the number of potential design configurations is very large. However, traditional approaches based on empirical data and phenomenological formulas suffer from poor accuracy. Moreover, such design practices focus on technology assessment using empirical relationships and historical data derived from systems developed previously (Kolonay 2013). However, many of the technologies and system configurations being evaluated have no historical or empirical information associated with them. Hence, the traditional assessment process produces inaccurate results, leading to ill-informed decisions.

The use of physics based high fidelity modeling has received considerable attention by the design community. Physics based modeling tools along with high end computing resources provide accurate multiphysics analysis and design in the early stages of design. High fidelity models that provide better accuracy are used in achieving an optimal design, but high fidelity modeling is complex and computationally intensive, often prohibitively so. Current research concerns judiciously moving some high fidelity analyses into the conceptual design phase. The growing challenges in conceptual design demand a platform to cope with the existing computational complexity, and the potential to reduce design cycle time and cost of production.

Aircraft analysis and design entail complex simulations, some I/O intensive, others requiring high floating point performance and high memory bandwidth. High performance computing (HPC) systems are critical for large scale design studies (Kodiyalam *et al*. 2004). While HPC systems deliver high computational power (capability computing) or high throughput (capacity computing), they are static resources with little scalability or flexibility. Service-oriented architecture (SOA) addresses the challenges faced by HPC systems in terms of scalability, availability, flexibility, and reliability. SOA not only incorporates the features of HPC systems, but

also promises a world of orchestrated services by creating dynamic processes and agile applications that span platforms and organizations (Georgakopoulous and Papazoglou 2008). An objective of this work is to carry out physics based MDO studies on a distributed computing platform that is robust, reliable, and cost effective.

1.2 SORCER

SORCER, a Java based network centric computing framework (maintained by SORCERsoft.com, a subsidiary of SMT S. A. group), is a federated service-to-service (S2S) metacomputing environment that treats service providers as network peers with well-defined semantics of a federated service object-oriented architecture (Sobolewski 2008a). SORCER provides a platform for high fidelity multidisciplinary design optimization, combining models from various disciplines into one integrated model. SORCER accommodates dynamic distribution of service providers and on-demand provisioning of resources, resulting in significant speedups and effective utilization of computational resources.

Relevant work on SORCER includes large scale design space exploration with three layers of converged programming languages for transdisciplinary computing (Sobolewski and Kolonay 2012a). From a software engineering point of view, SORCER's models are represented in a top-down var-oriented modeling language (VML) unified with programs in a bottom-up exertion-oriented language (EOL) (Sobolewski *et al*. 2013, Sobolewski and Kolonay 2012b). While VML accommodates computational fidelity within various types of evaluations, EOL describes engineering applications as a federation of local and remote services.

The Multidisciplinary Science and Technology Center at the United States Air Force Research Lab (AFRL) is using and developing SORCER to grapple with the computational complexity of physics based modeling in a distributed collaborative design environment (Kolonay 2013). Aerospace applications of SORCER include the design of the next generation efficient supersonic air vehicle (ESAV) described in (Burton *et al*. 2012), which employed the SORCER framework to automate multidisciplinary analysis (MDA) in a tightly integrated grid computing environment. The ability of SORCER to accommodate platform specific executables and integrate a variety of computing resources aided the MDA of an ESAV. The SORCER platform with three layers of converged programming supports dynamic fidelity for aeroelastic analysis and optimization. This capability facilitated the aeroelastic analysis with six different fidelities of induced drag in (Kolonay and Sobolewski 2011). SORCER is used as an integration environment for the comparison of four approximation techniques for highly nonlinear induced drag functions and their sensitivities with respect to control surface settings (Kolonay *et al*. 2008). The application of SORCER in the preliminary design of gas turbines is investigated in (Goel *et al*. 2005).

Other major players in the field of grid computing include the Globus Toolkit (Foster and Kesselman 1997), Condor (Thain *et al*. 2005), and Legion (Grimshaw and Wulf 1997). Grid middleware is the segment of the overall grid computing market that enables virtual organizations and the sharing of heterogeneous resources. The Globus Toolkit, Condor, and Legion can all be classified as major grid middleware. Globus is an open source software toolkit that facilitates construction of computational grids and grid based applications across institutional and geographic boundaries without sacrificing local autonomy. The Globus project involves research and development conducted by the Globus Alliance, which includes Argonne National Laboratory, Information Sciences Institute, and many others. Condor (name changed to HTCondor in September 2012) is an open source high-throughput computing software framework for coarse

grained distributed parallelization of computationally intensive tasks, developed and maintained by the HTCondor team at the University of Wisconsin, Madison. Legion is a vertically integrated object based metasystem that helps in combining large numbers of independently administered heterogeneous hosts, storage systems, database legacy codes, and user objects distributed over wide area networks (WAN) into a single, object based metacomputer that features a high degree of flexibility and site autonomy. Legion, developed and maintained by the University of Virginia, has been commercialized by Avaki.

While Globus and Legion can be classified as compute grids (cGrids), Condor belongs to a category of grids called metacompute grids (mcGrids). A compute grid is a virtual federation of processors that execute submitted executables with the help of a grid resource broker. A metacompute grid is a federation of service providers managed by a metacompute grid operating system. SORCER, on the other hand, belongs to a category of grids called intergrids (iGrids); an intergrid is a combination of a compute grid and a metacompute grid (Sobolewski 2008a).

The Federated Intelligent Product Environment (FIPER) is a mcGrid that was developed under the sponsorship of the National Institute for Standards and Technology (NIST). FIPER was built to form a federation of distributed services that provide engineering data, applications, and tools on a network. SORCER layers on top of FIPER a metacomputing OS with basic services, including a federated file system to support service-oriented metacomputing. The metacomputing environment along with a layer of abstraction (exertion-oriented programming) has been put to use in many grid computing projects including systems developed at GE Global Research Center, GE Aviation, and the AFRL.

1.3 Algorithms

VTDIRECT95, a massively parallel Fortran 95 implementation of D. R. Jones' algorithm DIRECT, is widely used in multidisciplinary design optimization, e.g., the design space exploration of a high speed civil transport (HSCT) (Baker *et al*. 2000a), for which the parallel implementation with load balancing techniques significantly reduced the design space exploration time (Baker *et al*. 2000b). Using polynomial response surface approximations for the MDO of an HSCT, DIRECT succeeded in finding the global optimum in every optimization (Baker *et al*. 1998). DIRECT was also used to solve an aircraft routing problem involving real terrain data (Bartholomew-Biggs *et al*. 2003). Related applications of DIRECT include the design of a slider air-bearing surface (ABS) (Zhu and Bogy 2002), transmitter placement optimization (He *et al*. 2004), gas pipeline optimization (Carter *et al*. 2001), cell cycle modeling (Panning *et al*. 2008, Zwolak *et al*. 2005), and molecular genetic mapping (Ljungberg *et al*. 2004). Engineering applications of VTDIRECT95 include global and local optimization of the kinematics of flapping wings (Ghommem *et al*. 2012), optimization of drag reduction on a circular cylinder (Mehmood *et al*. 2011), and nonconvex quadratic minimization with either box or integer constraints (Gao *et al*. 2013).

QNSTOP is a class of parallel quasi-Newton methods for stochastic optimization and deterministic global optimization. The application of QNSTOP to the global optimization of a 57-dimensional biomechanics model of human balance utilizing forward dynamic simulations is investigated in (Easterling *et al*. 2012). The application of QNSTOP to eukaryotic cell cycle modeling is discussed in (Andrew *et al*. 2014). QNSTOP for stochastic optimization problems synthesizes ideas from numerical optimization and response surface methodology, and demonstrates potential for stochastic robust design optimization and stochastic MDO problems.

The paper is organized as follows. Section 2 outlines the SORCER framework and the two optimization algorithms, VTDIRECT95 and QNSTOP. Section 3 presents details about conversion of VTDIRECT95 and QNSTOP to SORCER services. The description of EBF3PanelOpt, a framework for optimization of curvilinear blade-stiffened panels, is presented in Section 1 (Part 2). Section 2 (Part 2) includes results for optimization of curvilinear blade-stiffened panels using VTDIRECT95 and QNSTOP on a SORCER grid. The paper concludes in Section 3 (Part 2) with some suggestions for future work in this area of research.

2. Background

2.1 Overview of service-oriented computing and SORCER

Service-oriented computing is a computing paradigm that utilizes self-describing, platform-agnostic services as the fundamental constructs to support rapid, cost-effective composition of distributed applications (Papazoglou *et al*. 2007). Services are self-adapting, dynamic processes that effectively communicate with one another to perform user-requested tasks in a distributed computing environment. The service-oriented computing paradigm, derived from the SOA model, allows interoperability, reusability, and loose coupling of its components in a dynamic environment, where computer resources are assigned to services as and when necessary. As indicated in Fig. 1, the interaction between software agents is facilitated by message exchanges between service providers and service requestors. The service provider determines a description for a service and publishes it to a service discovery agency. This, in turn, is made discoverable to a service requestor. To invoke a service, the service requestor retrieves the service description from a registry and binds with the service provider based on the service description. In short, SOA addresses the challenges of distributed computing by enabling service discovery, integration, and use (Georgakopoulous and Papazoglou 2008).

SORCER is based on the concepts of SOA and also incorporates features of the service object-oriented architecture (SOOA), where service providers are objects accepting remote invocations (Sobolewski 2008a). As shown in Fig. 1, the service requestor binds to the service provider by creating a proxy for remote communication. SOOA permits great flexibility in terms of communication between agents. These proxies, known as smart proxies, grant access to local and remote resources, regardless of who initially created the proxy. In SORCER, providers broadcast their availability, registries intercept broadcasted announcements and cache proxy objects to their service providers (Sobolewski 2012). The SORCER operating system (SOS) looks up proxies by sending queries to registries and making selections from the available services. In short, providers use discovery/join protocols to publish services in the network, and SOS uses discovery/join protocols to obtain services in the network. From an object-oriented programming point of view, service providers are represented as independent network objects, locating each other via service registries and communicating through protocols such as remote method invocation (RMI), simple object access protocol (SOAP), common object request broker architecture (COBRA), etc.

Further, SORCER introduces three layers of converged programming abstractions: exertion-oriented programming (EOP), var-oriented programming (VOP), and var-oriented modeling (VOM) (Sobolewski and Kolonay 2011). The EOP abstraction manages object-oriented distributed system complexity introduced by the complex network of metacomputers. VOP is a paradigm based on dataflow principles where changing the value of a var automatically forces recalculation

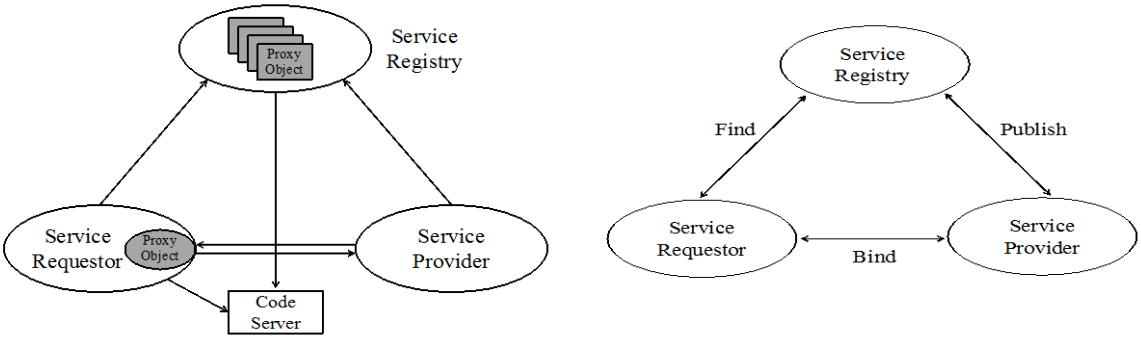

Fig. 1 Service-oriented architecture (SOA) (left) vs. service object-oriented architecture (SOOA) (right)

of the interdependent values of vars. VOM, a modeling paradigm using vars, defines heterogeneous multidisciplinary var-oriented models in large scale multidisciplinary models. Thus, the SORCER framework incorporates the power of object-oriented programming and exertion-oriented programming to create an infrastructure that is modular, extensible, and reusable. All of the above concepts are defined precisely and discussed in more detail in later sections.

Based on successful implementation of large scale engineering applications with SORCER (Kolonay 2013, Burton *et al*. 2012, Kolonay *et al*. 2008, Kolonay and Sobolewski 2011, Goel *et al*. 2005, Kolonay *et al*. 2007, Xu *et al*. 2008), this section outlines several desirable features related to design space exploration pertinent to multidisciplinary aircraft analysis and design optimization.

- Large scale, distributed, decentralized: SORCER dynamically federates processes and smartly distributes the load across all machines in the network, thereby resulting in significant reduction of design cycle time.
- Leveraging the power of HPC: SORCER provides the features and computing power of HPC and SOA to form a dynamic distributed engineering collaboration platform.
- Reusability: The incorporation of object-orient modularity enables a high level of reuse when moving from one study to the next.
- Cost effective: SORCER accommodates physics based modeling via HPC for faster evaluation of higher fidelity configurations at the preliminary level of design when compared to traditional practices.
- Better utilization of computational resources: SORCER enables collaborative design studies across organizational boundaries and maximum utilization of all compute resources on the network, ranging from personal computers to high performance computing machines.
- Distributed resource management: SORCER employs Jini Connection technology (now called Apache River) with its JavaSpaces service to implement computational resource management across the network. The JavaSpaces technology facilitates the implementation of a self-load limiting grid computing system that can dynamically grow and shrink during the course of an optimization study (Freeman *et al*. 1999). The loosely coupled space-based service federation allows asynchronous communication between computers in the network in a reliable manner (Sobolewski 2008b).

The above-mentioned features contribute to significantly accelerate design computations via

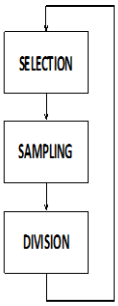

Fig. 2 Essential operations of the algorithm DIRECT

distribution of tasks in a network. In (Alyanak 2013), a sequential linear programming (SLP) algorithm to minimize the gross take-off weight (GTOW) of a vehicle is implemented on a SORCER grid. The SLP method is customized for taking advantage of SORCER's parallel computing capability such that gradient and line search calculations are executed in parallel. This methodology resulted in a reduction of the optimization time from 24 hours to two hours. Thus, the SORCER framework exhibits a wide range of capabilities that make large scale, multidisciplinary design studies feasible.

2.2 VTDIRECT95

VTDIRECT95 is a Fortran 95 software package using massively parallel dynamic data structures to implement the algorithm DIRECT by Jones *et al*. (1993). The algorithm DIRECT (DIviding RECTangles) is a deterministic global optimization algorithm that performs Lipschitzian optimization without the Lipschitz constant, and can be classified as a derivative free direct search algorithm. Each iteration of VTDIRECT95 consists of three essential operations as shown in Fig. 2: region selection (SELECTION), point sampling (SAMPLING), and space division (DIVISION).

Let E^n denote real n-dimensional Euclidean space, $D=\{x\in E^n|\ell\le x\le u\}$ be a box in E^n, and $f: D\to E$ a Lipschitz continuous function. The problem is to find a global minimum point $\overline{x}$ of f over D, $f(\overline{x}) = \min_{x\in D} f(x)$. The original (serial) algorithm by Jones *et al*. (1993) is described in six steps as below:

Step 1 (initialization): Normalize the feasible set D to be the unit hyper-cube. Sample the center point c_i of this hypercube and evaluate $f(c_i)$. Initialize $f_{\min}:=f(c_i)$ evaluation counter $m:=1$, and iteration counter $t:=0$.

Step 2 (selection): Identify the set S of "potentially optimal" boxes (subregions) of D. A box is potentially optimal if, for some Lipschitz constant, the function value within the box is potentially smaller than that in any other box (a formal definition with parameter ε is given by Jones *et al*. (1993)).

Step 3 (sampling): For any box $j\in S$, identify the set I of dimensions with the maximum side length. Let δ equal one-third of this maximum side length. Sample the function at the points $c\pm\delta e_i$ for all $i\in I$, where c is the center of the box and e_i is the ith unit vector.

Step 4 (division): Divide the box j containing c into thirds along the dimensions in I, starting with the dimension with the lowest value of $w_i=\min\{f(c+\delta e_i), f(c-\delta e_i)\}$, and continuing to the dimension with the highest wi. Update $f_{\min}$ and m.
Step 5 (iteration): Set $S:=S\backslash\{j\}$. If $S \neq \emptyset$, go to Step 3.
Step 6 (termination): Set $t:=t+1$. If iteration limit or evaluation limit has been reached, stop. Otherwise, go to Step 2.

VTDIRECT95 has numerous modifications from DIRECT in order to improve performance and load balancing on large scale parallel systems. The massively parallel implementation VTDIRECT95 distributes data among processors to share the memory burden imposed by storing all current boxes. Numerous hierarchical and fully distributed control schemes have been tried, with the most effective being that shown in Fig. 3. The parallel scheme for SELECTION concentrates on distributing data among multiple masters to share the memory burden. Functional parallelism for SAMPLING is achieved by fully distributed control allocating function evaluation tasks to workers.

On the top level, independent optimizations are done in the m subdomains (SDs). Within each subdomain, n subdomain masters (SMs) collaborate on SELECTION in parallel. On the bottom level, k workers (Ws) in a global pool request function evaluation tasks from all the subdomain masters to accomplish SAMPLING. SD_i denotes subdomain i, $SM_{i,j}$ denotes subdomain master j in SD_i, and W_k is worker k that works for all the SMs in active SDs. A detailed discussion of the implementation of the serial and parallel subroutines in VTDIRECT95 is presented in (He *et al*. 2009).

2.3 QNSTOP

QNSTOP is a class of quasi-Newton methods for stochastic optimization with variations for deterministic global optimization (Amos *et al*. 2014b). The Fortran 2003 implementation of QNSTOP consists of serial and parallel codes for the quasi-Newton stochastic optimization method of Castle and Trosset (Castle 2012). Both variations are described simultaneously in the following.

In iteration k, QNSTOP methods compute the gradient vector $\hat{g}_k$ and Hessian matrix $\hat{H}_k$ of a quadratic model

$$\hat{m}_k(X - X_k) = \hat{f}_k + \hat{g}_k^T(X - X_k) + \frac{1}{2}(X - X_k)^T \hat{H}_k(X - X_k)$$

of the objective function f centered at X_k, where $\hat{f}_k$ is generally not $f(X_k)$.

In the unconstrained context, QNSTOP methods progress by

$$X_{k+1} = X_k - [\hat{H}_k + \mu_k W_k]^{-1}\hat{g}_k ,$$

where μ_k is the Lagrange multiplier of a trust region subproblem and W_k is a scaling matrix. For constrained problems with feasible set Θ, the update is

$$X_{k+1} = [X_k - [\hat{H}_k + \mu_k W_k]^{-1}\hat{g}_k]_\Theta ,$$

where $[.]_\Theta$ denotes projection on the feasible set Θ.

2.3.1 Estimating the gradient

Regression experiments in QNSTOP are designed in a region of interest containing the current iterate. QNSTOP uses an ellipsoidal design region centered at the current iterate $X_k \in E^p$. Let

$$W_\gamma = \{W \in E^{p\times p} : W = W^T,\ \det(W) = 1,\ \gamma^{-1} I_p \preccurlyeq\ W \preccurlyeq\ \gamma I_p\}$$

for some $\gamma \geq 1$ where I_p is the $p \times p$ identity matrix. The shape of the ellipsoidal design regions with eccentricity constrained by γ is controlled by the valid scaling matrices represented by the elements of the set W_γ. Let the ellipsoidal design regions

$$E_k(\tau_k) = \{X \in E^p : (X - X_k)^T W_k (X - X_k) \leq \tau_k^2\},$$

where $W_k \in W_\gamma$. In the deterministic case, if there is no gain, $\tau_k = \tau_0 > 0$; otherwise, for gain $\zeta > 0$, let

$$\tau_k = \frac{\zeta}{\zeta + k}\tau_0.$$

In the stochastic case, the convergence theory entails that τ_k be decayed according to the formula $\tau_k = a(k+1)^{-b}$, where $a > 0$ and $b \in (0, 0.5)$.

In each iteration, QNSTOP methods choose a set of N_k design sites $\{X_{k1}, \ldots, X_{kN_k}\} \subset E_k(\tau_k) \cap \Theta$. In this implementation, $N = N_k$ is fixed for each $k = 1, 2, \ldots$ and $X_{k1}, \ldots, X_{kN} \in E_k(\tau_k) \cap \Theta$ are uniformly sampled in each iteration. Let $Y_k = (y_{k1}, \ldots, y_{kN})^T$ denote the N-vector of responses where $y_{ki} = F(X_{ki}) + \text{noise}$. The response surface is modeled by the linear model $y_{ki} = \hat{f}_k + X_{ki}^T \hat{g}_k + \varepsilon_{ki}$, where ε_{ki} accounts for the lack of fit. Let $\bar{X}_k = N^{-1} \sum_{i=1}^{N} X_{ki}$ and

$$D_k = \begin{bmatrix} (X_{k1} - \bar{X}_k)^T \\ \vdots \\ (X_{kN} - \bar{X}_k)^T \end{bmatrix}$$

be the absolute deviations of X_{ki}. The least squares estimate of the gradient $\hat{g}_k$, ignoring the estimate for $\hat{f}_k$, is obtained by observing the responses and solving

$$(D_k^T D_k)\hat{g}_k = D_k^T Y_k.$$

2.3.2 Updating the model Hessian matrix

In the stochastic context, QNSTOP methods constrain the Hessian matrix update to satisfy

$$-\eta I_p \preccurlyeq\ \hat{H}_k - \hat{H}_{k-1} \preccurlyeq\ \eta I_p$$

for some $\eta \geq 0$. Conceptually, this prevents the quadratic model from changing drastically from one iteration to the next. A variation of the SR1 (symmetric, rank one) update $\hat{H}_k$ that satisfies this constraint is computed. However, the constraint is simply relaxed in the deterministic case and the BFGS update is used.

2.3.3 Step length control

QNSTOP methods use an ellipsoidal trust region concentric with the design region for controlling step length. In the deterministic case, the trust region ellipsoidal radius ρ_k is considered to be equal to the design ellipsoidal radius τ_k, and the next iterate X_{k+1} is the solution to the optimization problem

$$\min_{X \in E_k(\rho_k)} \hat{g}_k^T (X - X_k) + \frac{1}{2}(X - X_k)^T \hat{H}_k (X - X_k).$$

In the stochastic case, the trust region ellipsoid radius ρ_k is different from the design ellipsoidal radius τ_k, and the next iterate

$$X_{k+1} = X_k - [\hat{H}_k + \mu_k W_k]^{-1} \hat{g}_k$$

is obtained by directly updating the Lagrange multiplier μ_k as described in Castle (2012). In both cases the computed point X_{k+1} is projected onto the feasible set Θ.

2.3.4 Updating the experimental design region

QNSTOP estimates an ellipsoidal confidence set, and uses this to update the scaling matrix W_k to W_{k+1} which then defines the next design region centered at X_{k+1}. The somewhat involved statistical details are given in Castle (2012) and Amos *et al*. (2014b).

2.3.5 Algorithm summary

In both modes of operation, global and stochastic, it is desirable to run QNSTOP from multiple start points. The algorithm outlined below is repeated for each start point.

Step 0 (initialization): Given a function evaluation budget $\tilde{B}$ per start point and operating mode (deterministic or stochastic), set values for $\tau_0>0$, $\mu_0>0$, $\gamma \geq 1$, $\eta \geq 0$, $\zeta \geq 0$, N, X_0, $k := 0$, $W_0 := \hat{H}_0 := I_p$.

Step 1 (regression experiment): Depending on the mode, compute τ_k. Uniformly sample $\{X_{k1}, \ldots, X_{kN}\} \subset E_k(\tau_k) \cap \Theta$. Observe the response vector $Y_k = (y_{k1}, \ldots, y_{kN})^T$. Compute $\hat{g}_k$.

Step 2 (secant update): If $k>0$, compute the model Hessian matrix $\hat{H}_k$ using BFGS (deterministic) or SR1 variant (stochastic) update.

Step 3 (update iterate): Compute μ_k depending on the mode as described in Section 2.3.3, solve $[\hat{H}_k + \mu_k W_k] s_k = -\hat{g}_k$ and compute $X_{k+1} = (X_k + s_k)_\Theta$.

Step 4 (update subsequent design ellipsoid): Compute an updated scaling matrix $W_{k+1} \in W\gamma$.

Step 5: If $(k+2)/(N+1)+1 < \tilde{B}$ then increment k by 1 go to Step 1. Otherwise, the algorithm terminates. (f is also observed at each ellipsoid center X_k.)

The algorithm QNSTOP has three significant sources of parallelism: the individual function evaluations, the loop over the samples in an experimental design, and the loop over the start points. A master-slave paradigm is a reasonable approach if the individual function evaluations are large scale parallel simulations. On large shared memory systems, ample parallelism is exhibited at the two outer nested loops-the loop over the start points and the loop over the samples $f(X_{ki})$ in an experimental design $\{X_{kl}\}_{i=1}^N$.

A detailed discussion of the serial and parallel implementations of QNSTOP can be found in (Amos *et al*. 2014b). An analysis of a serial Fortran 95 implementation of QNSTOP is presented in (Amos *et al*. 2014a).

2.4 Discussion

In the context of ever increasing parallelism, higher dimensions, and multidisciplinary design optimization, algorithms like VTDIRECT95 (for deterministic global optimization) and QNSTOP (for stochastic optimization) are excellent candidates for SORCER services. Objective function cost is one of the key parameters that affects the parallel performance under different parallel schemes. High parallel efficiency involves balancing communication overhead with the distribution of evaluation tasks for good load balancing (He *et al*. 2009a, 2009b). While SORCER has no control of the definition and granularity of the tasks, it *can* provide robust distributed parallelization and load balancing across computational resources, thus significantly speeding up the evaluation of objective functions in a dynamically scalable metacomputing environment.

3. VTDIRECT95 and QNSTOP as SORCER services

3.1 JNI wrappers for VTDIRECT95 and QNSTOP

The massive growth of the internet and the World Wide Web (WWW) led to the development of the Java programming language, a language particularly suited for client-server web applications. The power of Java lies in its platform-independent compiled byte code programs destined for distribution on the internet. Further, the Java Virtual Machine (JVM), an abstract computing machine implemented in the Java Runtime Environment (JRE), helps developers run a program in a wide variety of distributed environments. Java code is executed in a sandbox environment that prevents the code from accessing the other parts of the machine, hence ensuring security.

SORCER leverages the power of distributed computing through the use of Java interoperability, Jini, and web services (Sobolewski 2008a). While the benefits of using Java in distributed computing are well known, the adoption of Java as a language for numerical computing presents difficulties. Despite a significant improvement in performance of the JVM in the past few years, some obstacles still remain: over-restrictive floating point semantics, inefficient support for complex numbers and alternative arithmetic systems, and lack of direct support for true multidimensional arrays (Boisvert *et al*. 2001). Moreover, the task of manually converting existing code in Fortran to Java-based services is both daunting and expensive (Liang 1999).

The Fortran 95 implementations of optimization algorithms considered in this paper, VTDIRECT95 and QNSTOP, are superior in design and performance to their FORTRAN 77 counterparts. These implementations incorporate advanced features such as derived data types, pointers, dynamic memory allocation, array segments, vector subscripts, modules, etc. These features enabled design of dynamic data structures that flexibly organized the data on a single machine, effectively reduced the local data storage, and efficiently shared the data across multiple processors (He *et al*. 2009). While Fortran is effective for numerical computing, Java provides flexibility and scalability for dynamic grid-based network architectures. In order to cope with the heterogeneity imposed by various programming languages, Java wrappers for the existing legacy code have been implemented using the JNI (Java Native Interface) libraries.

The JNI is a powerful feature of the Java platform that lets developers utilize code written in other languages such as C, C++, and Fortran. The JNI is a two-way interface that allows Java applications to invoke native code and vice versa. The JNI is an interface that is supported by all

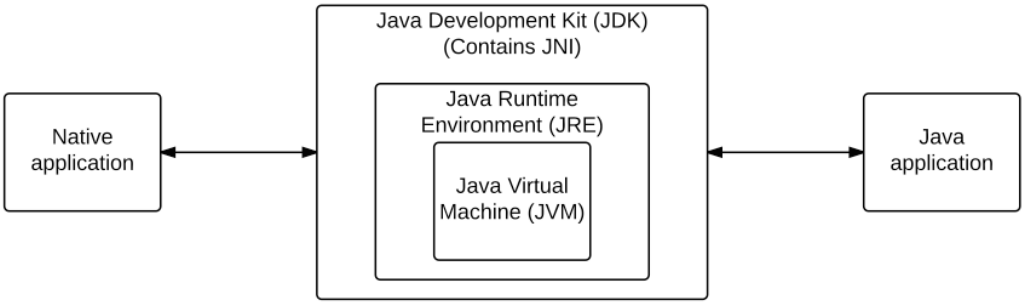

Fig. 4 Two-way interface provided by JNI

Java virtual machine implementations on a wide variety of host environments. One of the most important features of the JNI is the flexibility it offers-a single version of native code will run on different implementations of the JVM. Fig. 4 shows a block diagram that illustrates the two-way interface provided by JNI.

In developing the wrappers for the existing Fortran 95 implementations of VTDIRECT95 and QNSTOP, a feature of the JNI called the *invocation interface* was used. The invocation interface allows a regular non-Java program running on the native operating system to invoke a JVM to gain access to Java classes and features (Liang 1999). The invocation interface allows developers to embed a JVM implementation into native applications. Native applications can link with a native library that implements the JVM, and then use the invocation interface to execute components written in the Java programming language (Lindsey *et al*. 2010). Further, a C or C++ layer is required to gain access to codes written in Fortran. Such C or C++ code is often called the "glue code" since it is the *glue* that holds the Java and Fortran code together.

The motivation for using SORCER for MDO is the extensive exploration of the design space and the analyses of a large number of conceptual and preliminary design points during the early stages of design. In order to accomplish this, the design variables, objectives, and constraints should be made available during every iteration until the optimization algorithm converges to a global or local minimum point. In the SORCER environment, the objective function is implemented as a service, where the service is being requested by the optimization algorithm at every iteration to evaluate the objective function. The JNI wrapper acts as a layer of abstraction between the optimization algorithm and the Java block that evaluates the objective for a given design point. The block diagram illustrating the implementation of JNI wrappers is illustrated in Fig. 5.

3.2 VTDIRECT95 and QNSTOP as SORCER services

The concept of a service provider, or simply 'provider', is the crux of an engineering analysis or design study using SORCER. A provider is implemented in accordance with principles of exertion-oriented programming (EOP) and makes a number of services available to users in a distributed computing environment. EOP, a service-oriented programming paradigm using service providers and service commands, is a form of distributed programming that describes the distributed problem explicitly in terms of the intrinsically unpredictable network domain (Sobolewski and Kolonay 2012a). An *exertion* is an object that represents a process by specifying

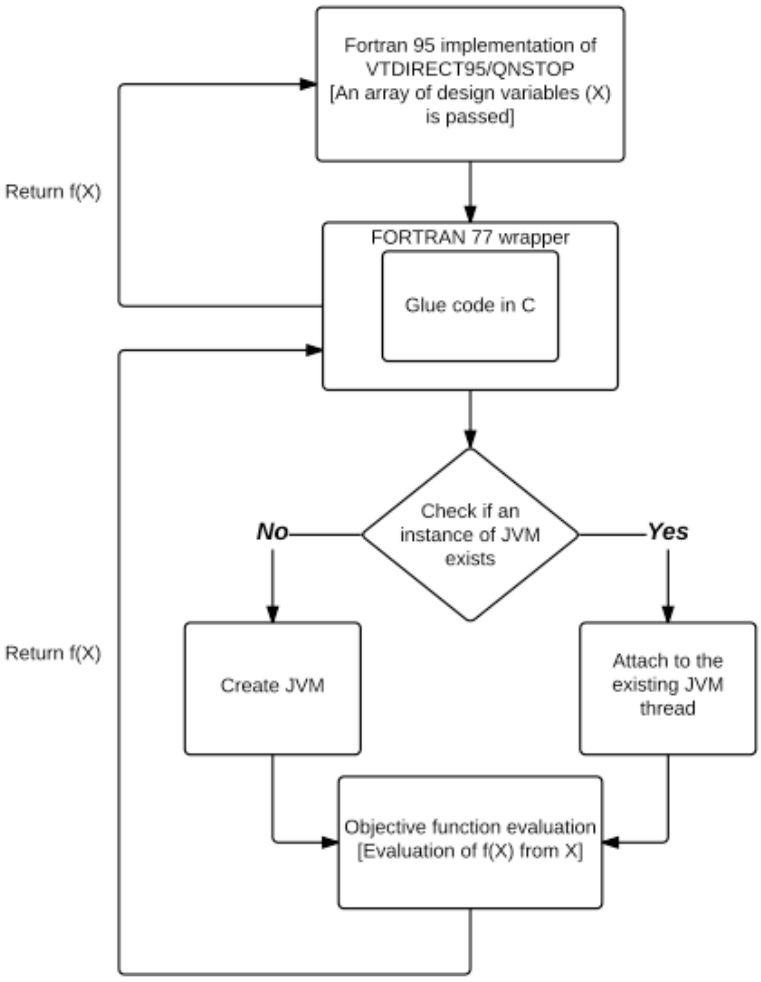

Fig. 5 Block diagram representing the objective function evaluation through JNI

the relationship between services and the information passed between them. In short, an exertion defines the collaboration between service-oriented programs. According to the central exertion principle, distributed processes are described by the interconnected federation of simple and effective service providers that compete with one another to be exerted. Service providers are network objects associated with leased network resources, and federate for executing a specific network request (exertion).

Programmatically, a provider is Java code that makes a number of Java methods (services) available to users over a network. A provider presents a Java interface to identify the service it provides. This Java interface is referred to as a *service type*. This approach of implementing providers not only provides a layer of abstraction from a specific implementation of a service, but also enforces polymorphism-multiple providers on the network may have different implementations of the same service but they would all implement the same service type (interface) (Burton *et al*. 2012).

A user may request a service by specifying the name of the service, the service type for that service, and the arguments for the service. An instance of the *Task* class is used to represent a basic unit of work. A 'task' is a service command for an individual request to be executed on a single service provider. The arguments for all SORCER services are instances of the *Context* class. A context object is a generic container that consists of several name-value pairs to specify input, or output, or both. In short, the input/output data associated with a task execution is called a context.

The provider is published on the network using SORCER. The provider's service may then be accessed via a small Java code called a *service requestor*. In short, an object that creates exertions and submits them to the grid is called a requestor; an object that accepts exertions from requestors and performs some calculations is called a provider.

Providers are of two kinds-*analysis providers* and *model providers*. Providers that leverage existing domain-specific codes are referred to as analysis providers (Burton *et al*. 2012). While the term 'analysis' typically refers to the process of solving a system of equations, an 'analysis provider' is an entity that neatly wraps the underlying domain-specific code with Java code so the domain-specific code can be accessed as a service by a remote user. The domain-specific code is generally platform independent and performs the bulk of the engineering-specific computations for a given service. A model provider is defined precisely in the following section.

3.3 Implementation of model provider for objective function evaluation

Design of complex systems requires a large number of dependent and independent variables. Values of the dependent variables are subject to several recalculations during the course of analysis. A change in the value of an independent variable does not necessarily force recalculation of a particular dependent variable. Further, the number of variables increases drastically with increasing complexity of the problem being solved. SORCER leverages the power of var-oriented programming (VOP) to handle large sets of interconnected variables. VOP is a programming paradigm using service-oriented variables called vars to design var-oriented multifidelity compositions (Sobolewski and Kolonay 2012a). A var is defined by a triplet < *value, evaluator,* f*ilter* >, where

- a *value* is an expression yielding a valid quantity;
- an *evaluator* defines the process of how data is produced via remote services, or produced locally;
- a *filter* reduces the data generated by the evaluator to the value of the var.

While VOP focuses on how evaluators calculate, a service-oriented modeling paradigm called var-oriented programming (VOM) focuses on how vars connect. VOM is a modeling paradigm using vars in a specific way to define large-scale analysis models such as response, parametric, and optimization models (Sobolewski and Kolonay 2011). In SORCER terminology, a *model* is a collection of vars.

In the context of optimization, these vars are the design variables and the implementation of the objective and constraint functions. Var instances are used to model both independent and dependent variables in SORCER. While independent vars are used as a container to store a value and perform no calculations, dependent vars implement mathematical functions. These dependent and independent vars that define a specific optimization problem are modeled as an instance of *OptimizationModel*. Such a model, when published on the network, is referred to as a *model provider*. The model provider is characterized by a single state and behaves like shared memory to users over the network. There are two ways for users to interact with a published model provider:

a) via a single model query; or b) via a table model query.

At each objective function evaluation, the communication between the optimizer (e.g., the Fortran 95 subroutines VTdirect or QNSTOPS) and the model provider is facilitated by instantiating the *ModelClient* class. The *ModelClient* class is instantiated in the JNI wrapper and provides a simple interface for setting design variable values and obtaining responses necessary to form the objective and constraint functions (Burton *et al*. 2012). For each objective function evaluation, a query object containing the name of the model provider, the design variable var names and values, and the var names of the objective function that the user wishes to calculate is constructed. When the query object is executed, the corresponding published model provider receives the query object and invokes the *setValue* method on all the design variables. Once the variables are assigned values, the model invokes the *getValue* method on the user-specified objective function. The query object is then returned to the user with the updated values.

In order to obtain the most recently updated value of the dependent var (the user-specified objective function), the model invokes the *evaluator* instance to check if the value of the var has changed since the last invocation of *evaluate*. The evaluator in turn calls the *getValue* method on its argument vars to ensure their respective values are current before proceeding. If the argument vars are current and unchanged, the evaluator returns the dependent var value without further processing. This demand driven aspect of EOP ensures that calls to *evaluate* be made only if the evaluator's arguments have changed since the last invocation of *evaluate*. Hence, change in value of any argument var automatically forces recalculation of the dependent var's value. Once the *evaluate* method gets a new value, an instance of the *filter* class is employed to pass the return value to an instance of the *persister* class, which in turn assigns the value to an object (Burton *et al*. 2012). At every objective function evaluation, the value is returned to glue code in C, which in turn returns the value to the program carrying out the optimization.

3.4 Serial subroutines VTdirect and QNSTOPS as SORCER services

For the serial implementations (the subroutines VTdirect and QNSTOPS) of the algorithms, platform independent executables are implemented using JNI as described in Section 3.1. Next, each individual executable is tightly coupled with the provider's service. As an example, the structure of the EO program for the serial VTdirect is:

```
// Create NetSignature
String providerName = Sorcer.getActualName("Engineering-
VTdirect"); String serviceName = "execute";
NetSignature methodEN = new NetSignature(serviceName, VTdirect.class,
        providerName);

// Create component exertion
NetTask vtdirectTask = new NetTask("run execute", "Task to run
VTdirect", methodEN);

// Create context
VTdirectContext context = new VTdirectContext("VTdirectContext");
context.setInputFile(vtdirectInputUrl);
context.setInputModelFile(modelInputUrl);
```

```
vtdirectTask.setContext(context);

// Exert collaboration
Exertion result = vtdirectTask.exert();
```

In the above EO program, a *signature* is defined by the name of the provider, the name of the interface, and the operation name used by any remote object to run the service. A task is defined by the name of the operation to be executed by a service provider. The input arguments to VTdirect are represented by the Context. The corresponding script that calls VTdirect is executed when the service composition (exertion) binds at runtime to the corresponding service provider.

The objective function is implemented as a model provider. For the serial subroutines VTdirect and QNSTOPS to avail themselves of required services (namely, objective function evaluations), the JNI wrapper interacts with the model provider via a single model query. At each objective function evaluation, the JNI wrapper constructs a query object containing the name of the model provider, the design variable var names and values, and the var names of the objective function that the user wishes to calculate. Once the query is executed, the model provider begins to *setValue* and *getValue* on the vars as described in Section 3.3. The objective function evaluation is carried out sequentially, and at every function evaluation, the value in the object returned by the model provider is parsed and returned to the program carrying out the optimization.

3.5 Parallel subroutines pVTdirect and QNSTOPP as SORCER services

3.5.1 SORCER and existing parallel optimization codes

In addition to (the serial subroutine) VTdirect as a SORCER service discussed in Section 3.4, it was intended to provide access to (the parallel subroutine) pVTdirect as a service. Unfortunately, parallel results for pVTdirect (the massively parallel implementation of DIRECT in the package VTDIRECT95) under SORCER are not presented here, because pVTdirect is fundamentally incompatible with efficient usage of the SORCER/JavaSpace/table model query paradigm implemented for this work (JavaSpace is described later in Section 4.2), on the hardware used for this work. An explanation of this statement follows. Of the several available paradigms for using SORCER, the most general and robust is the SORCER/JavaSpace/table model query paradigm, which is why this one was chosen here. This SORCER/JavaSpace/table model query paradigm tacitly assumes a master-slave parallel computing paradigm, and achieves its parallelism by chunking (binning) the function evaluations in function evaluation service calls to SORCER. This assumes that concurrent function evaluation points are all known at the same time (synchronization point), and that all these points are readily accessible by the master. These assumptions are valid for an optimization algorithm using a master-slave paradigm, such as QNSTOPP. They are not valid for a fully distributed algorithm such as pVTdirect, which both massively distributes all the data (the function evaluation points) and is asynchronous (the evaluation points in each iteration are *not* known at any synchronization point). In fact, these properties (fully distributed control, distributed data, and asynchrony) are precisely the reason for the massive scalability of pVTdirect. Indeed, a parallel master-slave version of DIRECT, compatible with the SORCER/JavaSpace/table model query paradigm, could be constructed, but doing so would vitiate all the desirable properties (fully distributed control and data, asynchrony, and the resulting scalability) of the sophisticated production code pVTdirect.

The table could have had just one row, which would then work with pVTdirect via MPI, but the

combined overhead of MPI and SORCER results in a parallel slowdown for the test problems and hardware used here, and hence this (single table row corresponding to a single function evaluation point) was not pursued further. With sufficient hardware (cores to support all the MPI and SORCER threads) and sufficiently expensive function evaluations, pVTdirect with SORCER would demonstrate parallel speedup.

3.5.2 Modifying parallel subroutine QNSTOPP for SORCER

The parallel (OpenMP) implementation (subroutine QNSTOPP) of QNSTOP incorporates three sources of parallelism: (1) the loop over the start points (of size NSTART), and (2) the loop over the experimental design samples $i=(1,\ldots,N)$, or (3) both. For compatibility with SORCER, QNSTOPP is modified at the level of the inner loop over the experimental design samples such that the function evaluations are chunked in function evaluation calls to SORCER. In this case, QNSTOPP interacts with the published model provider via a table model query. Rather than passing a single design point to the model provider, the JNI wrapper constructs a table containing the name of the design vars and their values for a set of sample points. As with the case of a single model query, a query object containing the containing the name of the model provider, the design variable var names and values, and the var names of the objective function is constructed. The model provider, on receiving the query object, creates new child instances for each row in the table for parallel execution of the table row evaluations. The model provider creates a thread for each child instance and begins to *setValue* and *getValue* on the vars. On completion, the var values for each run are then returned to the JNI wrapper in a table object and the child models are discarded.

This completes the background on SORCER, the algorithms DIRECT and QNSTOP, and the details of how those algorithms can be used by SORCER. Part 2 gives performance results for SORCER on several wing panel design problems, and draws some general conclusions about the suitability of SORCER for MDO involving the parallel optimization codes pVTdirect and QNSTOPP.

Acknowledgements

This material is based on research sponsored by Air Force Research Laboratory under agreement number FA8650-09-2-3938. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Laboratory or the U.S. Government. The authors thank Scott A. Burton for help with the SORCER installation, service implementation, and experiments.

References

Alyanak, E. (2013), *Multidisciplinary Design and Optimization of Efficient Supersonic Air Vehicles*, FY13 Scientific Advisory Board S & T Quality Review Presentation.

Amos, B.D., Easterling, D.R., Watson, L.T., Castle, B.S., Trosset, M.W. and Thacker, W.I. (2014a), "Fortran 95 implementation of QNSTOP for global and stochastic optimization", *Proceedings of the 22nd High Performance Computing Symposium (HPC 2014)*, Tampa, Florida.

Amos, B.D., Easterling, D.R., Watson, L.T., Thacker, W.I., Castle, B.S. and Trosset, M.W. (2014b), "Algorithm XXX: QNSTOP: quasi-newton algorithm for stochastic optimization", Computer Science Technical Report 2014-07, Virginia Polytechnic Institute and State University, Blacksburg, VA.

Andrew, T.M., Amos, B.D., Easterling, D.R., Oguz, C., Baumann, W.T., Tyson, T.T. and Watson, L.T. (2014), "Global parameter estimation for a eukaryotic cell cycle model in systems biology", *Proceedings of the Summer Simulation MultiConference*, Monterey, CA.

Baker, C.A., Grossman, B., Haftka, R.T., Mason, W.H. and Watson, L.T. (1998), "HSCT configuration design space exploration using aerodynamic response surface approximations", *Proceedings of the AIAA 98-4803, 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Saint Louis, MO.

Baker, C.A., Watson, L.T., Grossman, B., Haftka, R.T. and Mason, W.H. (2000a), "Study of a global design space exploration method for aerospace vehicles", *Proceedings of the AIAA 2000-4763, 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Long Beach, CA.

Baker, C.A., Watson, L.T., Grossman, B., Mason, W.H. and Haftka, R.T. (2000b), "Parallel global aircraft configuration design space exploration", Computer Science Technical Report TR-00-07, Virginia Polytechnic Institute and State University, Blacksburg, VA.

Bartholomew-Biggs, M.C., Parkhurst, S.C. and Wilson, S.P. (2003), "Global optimization approaches to an aircraft routing problem", *Euro. J. Operat. Res.*, **146**(2), 417-431.

Boisvert, R.F., Moreira, J., Philippsen, M. and Pozo, R. (2001), "Java and numerical computing", *IEEE Comput. Sci. Eng.*, **3**(2), 18-24.

Burton, S.A., Alyanak, E.J. and Kolonay, R.M. (2012), "Efficient supersonic air vehicle analysis and optimization implementation using SORCER", *Proceedings of the AIAA 2012-5520, 12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference* and *14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Indianapolis, Indiana.

Carter, R.G., Gablonsky, J.M. Patrick, A., Kelly, C.T. and Eslinger, O.J. (2001), "Algorithms for noisy problems in gas transmission pipeline optimization", *Optim. Eng.*, **2**(2), 139-157.

Castle, B.S. (2012), "Quasi-newton methods for stochastic optimization and proximity-based methods for disparate information fusion", Ph.D. Thesis, Indiana University, Bloomington, IN.

Easterling, D.R., Watson, L.T., Madigan, M.L., Castle, B.S. and Trosset, M.W. (2012), "Direct search and stochastic optimization applied to two nonconvex nonsmooth problems", *Proceedings of the 20th High Performance Computing Symposium*, Orlando, FL.

Foster, I. and Kesselman, C. (1997), "Globus: a metacomputing infrastructure toolkit", *Int. J. Supercomput. Appl.*, **11**(2), 115-128.

Freeman, E., Hupfer, S. and Arnold, K. (1999), *JavaSpaces Principles, Patterns, and Practice*, Addison Wesley Longman, Inc, Boston, MA.

Gao, D.Y., Watson, L.T. and Easterling, D.R. (2013), "Solving the canonical dual of box- and integer-constrained nonconvex quadratic programs via a deterministic direct search algorithm", *Optim. Meth. Softw.*, **28**(2), 313-326.

Georgakopoulous, D. and Papazoglou, M.P. (2008), *Service-Oriented Computing*, The MIT Press, Cambridge.

Ghommem, M., Hajj, M.R., Stanford, B.K., Watson, L.T. and Beran, P.S. (2012), "Global and local optimization of flapping kinematics", *Proceedings of the AIAA 2012-1983, 53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Honolulu, Hawaii.

Goel, S., Talya, S. and Sobolewski, M. (2005), "Preliminary design using distributed service-based computing", *Proceedings of the 12th ISPE International Conference on Concurrent Engineering*, Fort Worth, Texas.

Grimshaw, A.S. and Wulf, W.A. (1997), "The legion vision of a worldwide virtual computer", *Commun. ACM*, **40**(1), 39-45.

He, J., Verstak, A., Sosonkina, M. and Watson, L.T. (2009a), "Performance modeling and analysis of a massively parallel DIRECT: part 2", *Int. J. High Perform. Comput. Appl.*, **23**(1), 29-41.

He, J., Verstak, A., Watson, L.T. and Sosonkina, M. (2009b), "Performance modeling and analysis of a massively parallel DIRECT: part 1", *Int. J. High Perform. Comput. Appl.*, **23**(1), 14-28.

He, J., Verstak, A.A., Watson, L.T., Stinson, C.A., Ramakrishnan, N., Shaffer, C.A., Rappaport, T.S., Anderson, C.R., Bae, K.K., Jiang, J. and Tranter, W.H. (2004), "Globally optimal transmitter placement for indoor wireless communication systems", *IEEE Tran. Wireless Commun.*, **3**(6), 1906-1911.

He, J., Watson, L.T. and Sosonkina, M. (2009), "Algorithm 897: VTDIRECT95: serial and parallel codes for the global optimization algorithm DIRECT", *ACM Tran. Math. Softw.*, **36**(3), Article No. 17.

Jones, D.R., Perttunen, C.D. and Stuckman, B.E. (1993), "Lipschitzian optimization without the Lipschitz constant", *J. Optim. Theor. Appl.*, **79**(1), 157-181.

Kodiyalam, S., Yang, R.J., Gu, L. and Tho, C.H. (2004), "Multidisciplinary design optimization of a vehicle system in a scalable, high performance computing environment", *Struct. Multidiscip. Optim.*, **26**(3/4), 256-263.

Kolonay, R.M. (2013), "Physics-based distributed collaborative design for aerospace vehicle development and technology assessment", *Proceedings of the 20th ISPE International Conference on Concurrent Engineering*, Melbourne, Australia.

Kolonay, R.M. and Sobolewski, M. (2011), "Service ORiented Computing EnviRonment (SORCER) for large scale, distributed dynamic fidelity aeroelastic analysis and optimization", *Proceedings of the International Forum on Aeroelasticity and Structural Dynamics*, Paris, France.

Kolonay, R.M., Roberts, R.W. and Lambe, L.A. (2008), "A comparison of four approximation techniques for an Euler based induced drag function", *Proceedings of the AIAA 2008-5801, 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Victoria, British Columbia, Canada.

Kolonay, R.M., Thompson, E D., Camberos, J.A. and Eastep, F. (2007), "Active control of transpiration boundary conditions for drag minimization with an Euler CFD solver", *Proceedings of the AIAA 2007-*1891, *48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Honolulu, Hawaii.

Liang, S. (1999), *The Java Native Interface: Programmer's Guide and Specification*, Addison Wesley Longman Inc, MA.

Lindsey, C.S., Tolliver, J.S. and Lindblad, T. (2010), *JavaTech, an Introduction to Scientific and Technical Computing with Java*, Cambridge University Press, Cambridge, England.

Ljungberg, K., Holmgren, S. and Carlborg, O. (2004), "Simultaneous search for multiple QTL using the global optimization algorithm DIRECT", *Bioinformat.*, **20**(12), 1887-1895.

Mehmood, A., Akhtar, I., Ghommem, M., Hajj, M.R. and Watson, L.T. (2011), "Optimization of drag reduction on a cylinder undergoing rotary oscillations", *Proceedings of the AIAA 2011-1997, 52nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Denver, Colorado.

Panning, T.D., Watson, L.T., Allen, N.A., Chen, K.C., Shaffer, C.A. and Tyson, J.J. (2008), "Deterministic parallel global parameter estimation for a model of the budding yeast cell cycle", *J. Global Optim.*, **409**(4), 719-738.

Papazoglou, M.P., Traverso, P., Dustdar, S. and Leymann, F. (2007), "Service-oriented computing: state of the art and research challenges", *Comput.*, **40**(11), 38-45.

Raymer, D.P. (2006), *Aircraft Design: A Conceptual Approach*, AIAA Education Series, New York, NY.

Sobolewski, M. (2008a), "SORCER: computing and metacomputing intergrid", *Proceedings of the 10th International Conference on Enterprise Information Systems*, Barcelona, Spain.

Sobolewski, M. (2008b), "Federated collaborations with exertions", *Proceedings of the 17h IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE)*, Rome, Italy.

Sobolewski, M. (2012), *Object-Oriented Service Clouds for Transdisciplinary Computing, Cloud Computing and Services Sciences*, Springer.

Sobolewski, M. and Kolonay, R.M. (2011), "The convergence of three languages for transdisciplinary computing", available online at http://sorcersoft.org/publications/papers/2011/ce2011.pdf.

Sobolewski, M. and Kolonay, R.M. (2012a), "Service-oriented programming for design space exploration", *Proceedings of the 19th ISPE International Conference on Concurrent Engineering*, Trier, Germany.

Sobolewski, M. and Kolonay, R.M. (2012b), "Unified mogramming with var-oriented modeling and exertion-oriented programming languages", *Int. J. Commun. Netw. Syst. Sci.*, **5**(9A), 579-592.

Sobolewski, M., Burton, S. and Kolonay, R.M. (2013), "Parametric mogramming with var-oriented modeling and exertion-oriented programming languages", *Proceedings of the 20th ISPE International Conference on Concurrent Engineering*, Melbourne, Australia.

Thain, D., Tannenbaum, T. and Livny, M. (2005), "Distributed computing in practice: the condor experience", *J. Concurr. Comput. Pract. Exp.*, **17**(2-4), 323-356.

Xu, W., Cha, J. and Sobolewski, M. (2008), "A service-oriented collaborative design platform for concurrent engineering", *Adv. Mater. Res.*, **44-46**, 717-724.

Zhu, H. and Bogy, D.B. (2002), "DIRECT algorithm and its application to slider air-bearing surface optimization", *IEEE Tran. Magnet.*, **38**(5), 2168-2170.

Zwolak, J.W., Tyson, J.J. and Watson, L.T. (2005), "Parameter estimation for a mathematical model of the cell cycle in frog eggs", *J. Comput. Biol.*, **12**(1), 48-63.

EC