SERVICE ORIENTED COMPUTING ENVIRONMENT (SORCER) FOR DETERMINISTIC GLOBAL AND STOCHASTIC OPTIMIZATION

Chaitra Raghunath

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science and Applications

Layne T. Watson, Chair
Rakesh K. Kapania
Raymond M. Kolonay
Clifford A. Shaffer

July 10, 2015
Blacksburg, Virginia

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

SERVICE ORIENTED COMPUTING ENVIRONMENT (SORCER) FOR DETERMINISTIC GLOBAL AND STOCHASTIC OPTIMIZATION

Chaitra Raghunath

(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. Results are included for an aircraft design application.

ACKNOWLEDGEMENTS

To begin with, I would like to express my heartfelt gratitude to my advisor, Dr. Layne Watson for his scholarly advice, abundant patience, and tremendous encouragement, which helped steer my research in the right direction. I would like to thank Dr. Rakesh Kapania for letting me be a part of the SpaRibs group and for providing feedback on my research. I would like to thank Dr. Raymond Kolonay for his timely help with the SORCER software and for always making time for discussions. I am thankful to Dr. Clifford Shaffer for being a part of my committee and for providing valuable feedback on my thesis.

I am thankful to my colleagues, Nathan Love and Mohamed Jrad, for devoting quality time and energy to guide me in the final few stages of my research. I am very grateful to Dr. Scott Burton for being a friend and a mentor, and for providing timely help with the SORCER software. I would like to thank my labmates, Rishu Saxena and Sally Hamouda, for being very supportive and encouraging at all times. I am thankful to Rob Hunter and Steve Edwards for their administrative help.

Last but not the least, I would like to thank my family and friends for their constant support and undying faith in my abilities. I would especially like to thank my parents, my sister, and my brother-in-law, for their tremendous encouragement and unwavering love towards me.

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 EBF3PanelOpt code was developed under a research contract from NASA Fundamental Aeronautics Program to Virginia Polytechnic Institute and State University with Karen M. B. Taminger as the Program Manager.

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

Chapter 1.

INTRODUCTION

This thesis 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 [43]. 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 [28]. 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 [32]. 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 [17]. 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 [44]. 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 [48]. 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) ([47], [49]). 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 [28]. Aerospace applications of SORCER include the design of the next generation efficient supersonic air vehicle (ESAV) described in [5], 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 [30]. 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 [29]. The application of SORCER in the preliminary design of gas turbines is investigated in [18].

Other major players in the field of grid computing include the Globus Toolkit [15], Condor [52], and Legion [19]. 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 [44].

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) [8], for which the parallel implementation with load balancing techniques significantly reduced the design space exploration time [9]. Using polynomial response surface approximations for the MDO of an HSCT, DIRECT succeeded in finding the global optimum in every optimization [6]. DIRECT was also used to solve an aircraft routing problem involving real terrain data [7]. Related applications of DIRECT include the design of a slider air-bearing surface (ABS) [54], transmitter placement optimization [23], gas pipeline optimization [12], cell cycle modeling ([42], [55]), and molecular genetic mapping [36]. Engineering applications of VTDIRECT95 include global and local

optimization of the kinematics of flapping wings [16], optimization of drag reduction on a circular cylinder [38], and nonconvex quadratic minimization with either box or integer constraints [20].

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 [13]. The application of QNSTOP to eukaryotic cell cycle modeling is discussed in [2]. 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 thesis is organized as follows. Chapter 2 outlines the SORCER framework and the two optimization algorithms, VTDIRECT95 and QNSTOP. Chapter 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 Chapter 4. Chapter 5 includes results for optimization of curvilinear blade-stiffened panels using VTDIRECT95 and QNSTOP on a SORCER grid. The thesis concludes in Chapter 6 with some suggestions for future work in this area of research.

Chapter 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 [41]. 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 Figure 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 [17].

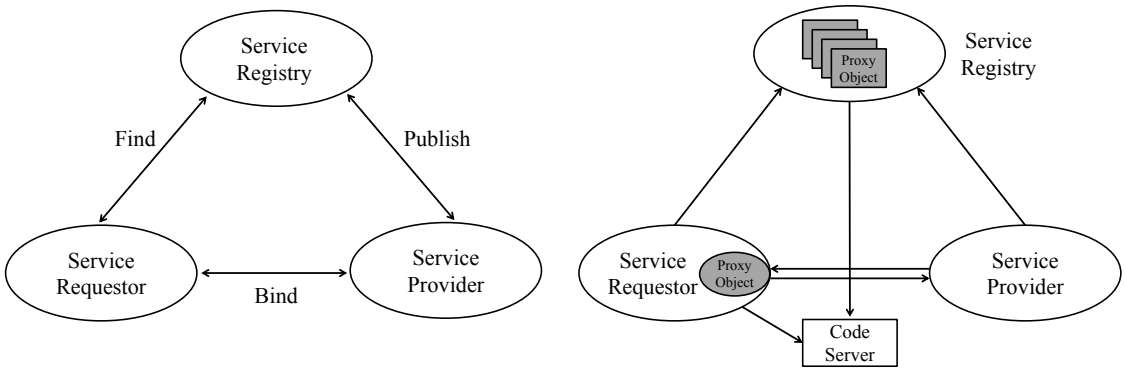


Figure 1. Service-oriented architecture (SOA) (left) vs. service object-oriented architecture (SOOA) (right).

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 [44]. As shown in Figure 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 [45]. 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) [50]. 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 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 chapters.

Based on successful implementation of large scale engineering applications with SORCER ([28], [5]–[18], [31], [53]), this chapter 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 drastic 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 [14]. The loosely coupled space-based service federation allows asynchronous communication between computers in the network in a reliable manner [46].

The above mentioned features contribute to significantly accelerate design computations via distribution of tasks in a network. In [1], 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. [25]. The algorithm DIRECT (DIviding RECTangles) is a deterministic global optimization algorithm that performs Lipschitzian optimization without the Lipschitz constant, and can be classified as

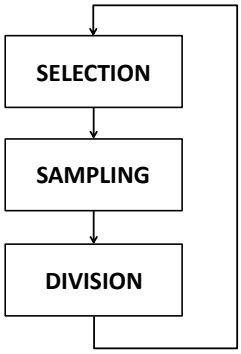


Figure 2. Essential operations of the algorithm DIRECT.

a derivative free direct search algorithm. Each iteration of VTDIRECT95 consists of three essential operations as shown in Figure 2: region selection (SELECTION), point sampling (SAMPLING), and space division (DIVISION).

Let E^n denote real n-dimensional Euclidean space, $D = \left\{x \in E^n \mid \ell \leq x \leq u\right\}$ be a box in E^n, and $f : D \to E$ a Lipschitz continuous function. The problem is to find a global minimum point $\bar{x}$ of f over D, $f(\bar{x}) = \min\limits_{x \in D} f(x)$. The original (serial) algorithm by Jones et al. [25] is described in six steps as below:

Step 1 (initialization): Normalize the feasible set D to be the unit hypercube. 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. [25]).

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 w_i. Update f_{min} and m.

Step 5 (iteration): Set $S := S \setminus \{j\}$. 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 Figure 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.

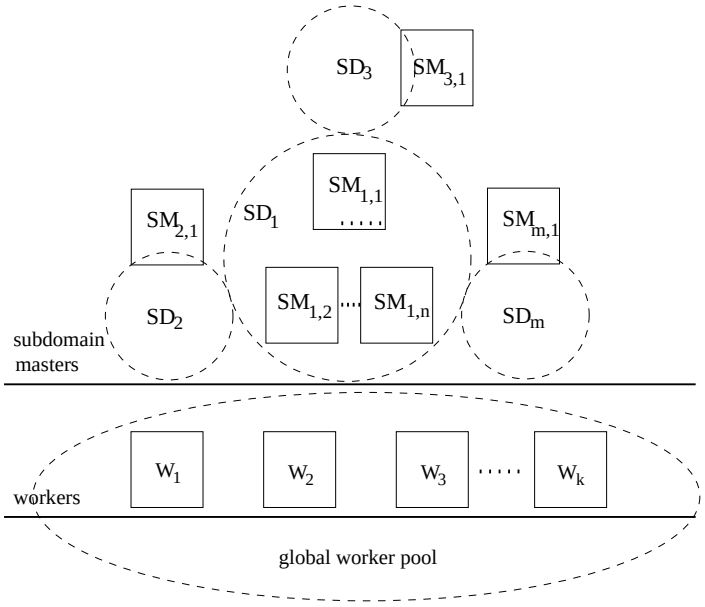


Figure 3. The parallel scheme of VTDIRECT95.

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 [24].

2.3. QNSTOP

QNSTOP is a class of quasi-Newton methods for stochastic optimization with variations for deterministic global optimization [4]. The Fortran 2003 implementation of QNSTOP consists of serial and parallel codes for the quasi-Newton stochastic optimization method of Castle and Trosset [11]. 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

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

where $[\cdot]_\Theta$ 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

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

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

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

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 $\{X_{k1}, ..., X_{kN_k}\} \subset E_k(\tau_k) \cap \Theta$. 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 $y_{ki} = \hat{f}_k + X_{ki}^T \hat{g}_k + \epsilon_{ki}$, 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

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

2.3.2 Updating the Model Hessian Matrix. In the stochastic context, QNSTOP methods constrain the Hessian matrix update to satisfy

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

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

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

is obtained by directly updating the Lagrange multiplier μ_k as described in Castle [11]İn 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 [11]and Amos et al. [4].

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 $\widetilde{B}$ 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 Chapter 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 $(k+2)(N+1)+1 < \widetilde{B}$ 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 $\left\{X_{ki}\right\}_{i=1}^{N}$.

A detailed discussion of the serial and parallel implementations of QNSTOP can be found in [4]. An analysis of a serial Fortran 95 implementation of QNSTOP is presented in [3].

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 ([21], [22]). 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.

Chapter 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 [44]. 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 drastic improvement in performance of the JVM in the past few years, some obstacles still remain: overrestrictive floating point semantics, inefficient support for complex numbers and alternative arithmetic systems, and lack of direct support for true multidimensional arrays [10]. Moreover, the task of manually converting existing code in Fortran to Java-based services is both daunting and expensive [33].

The Fortran 95 implementations of optimization algorithms considered in this thesis, 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 [24]. 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

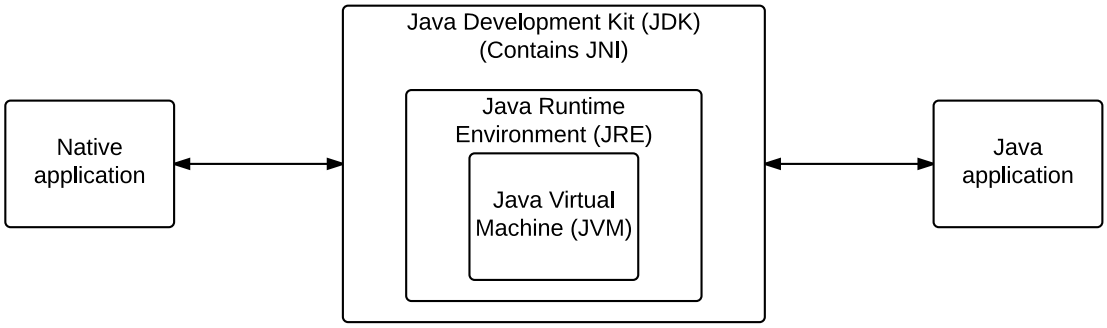


Figure 4. Two-way interface provided by JNI.

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 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. Figure 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 [33]. 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 [34]. 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.

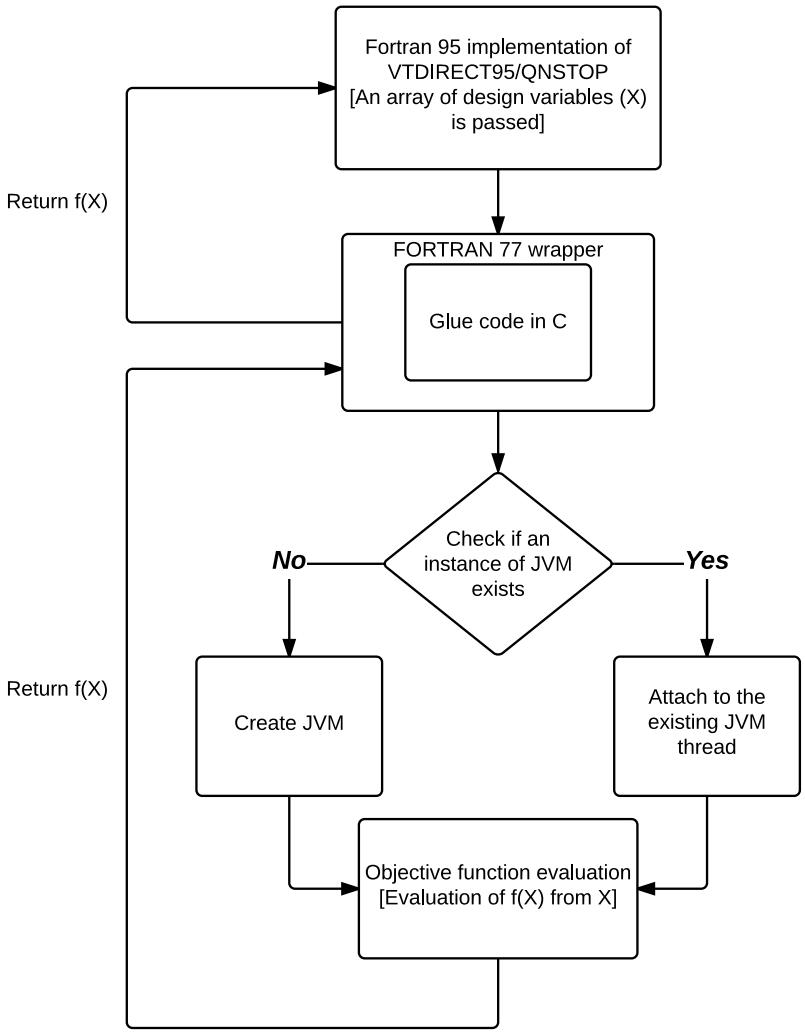


Figure 5. Block diagram representing the objective function evaluation through JNI.

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 Figure 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 [48]. An *exertion* is an object that represents a process by specifying 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) [5].

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 [5]. 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 chapter.

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 [48]. A var is defined by a triplet $\langle$*value, evaluator, filter* $\rangle$, 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 [50]. 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 [5]. 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 [5]. 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 Chapter 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 Chapter 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 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 Chapter 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 Chapter 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 ($i = 1, ..., N$), 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.

Chapter 4.

OPTIMIZATION OF CURVILINEAR BLADE-STIFFENED PANELS

4.1. Problem Description

Advances in manufacturing technology, computational science, and material science have produced a new generation of custom built so-called unitized structures that have multifunctionality tailored to design requirements. With additive manufacturing technology such as electron beam free-form fabrication [51], it is possible to produce arbitrary curved metallic structures using aerospace alloys like titanium and aluminum. The possibility of curved stiffening members enlarges the design space and leads to the possibility of a more efficient aircraft design. Previous research ([39]–[27]) has shown that curvilinear stiffeners can improve the buckling resistance of local panels. Locatelli et al. [37] demonstrated a savings in weight from the structural optimization of an aircraft wing using curvilinear spars and ribs (SpaRibs). The aircraft wing can be decomposed into multiple local panels bordered with the spars and ribs. Therefore, minimizing the structural weight of these panels reduces the overall wing weight.

Powerful computational environments have been developed in order to make use of such flexibility and build optimal light weight structures ([40], [35]). The framework *EBF3PanelOpt* described here (Figure 6) facilitates the structural optimization of curvilinearly stiffened panels by considering a number of constraints that have to be satisfied (buckling, von Mises stress, and crippling constraints). The framework, written in the scripting language Python, interacts with the commercial software MSC Patran (for geometry and mesh creation) and MSC Nastran (for finite element analysis). Given the input parameters and design variables, the script then creates the appropriate session file and submits it to MSC Patran to create the geometry and mesh of the stiffened panel, with which MSC Nastran then carries out a finite element analysis producing structural mass, buckling factor, von

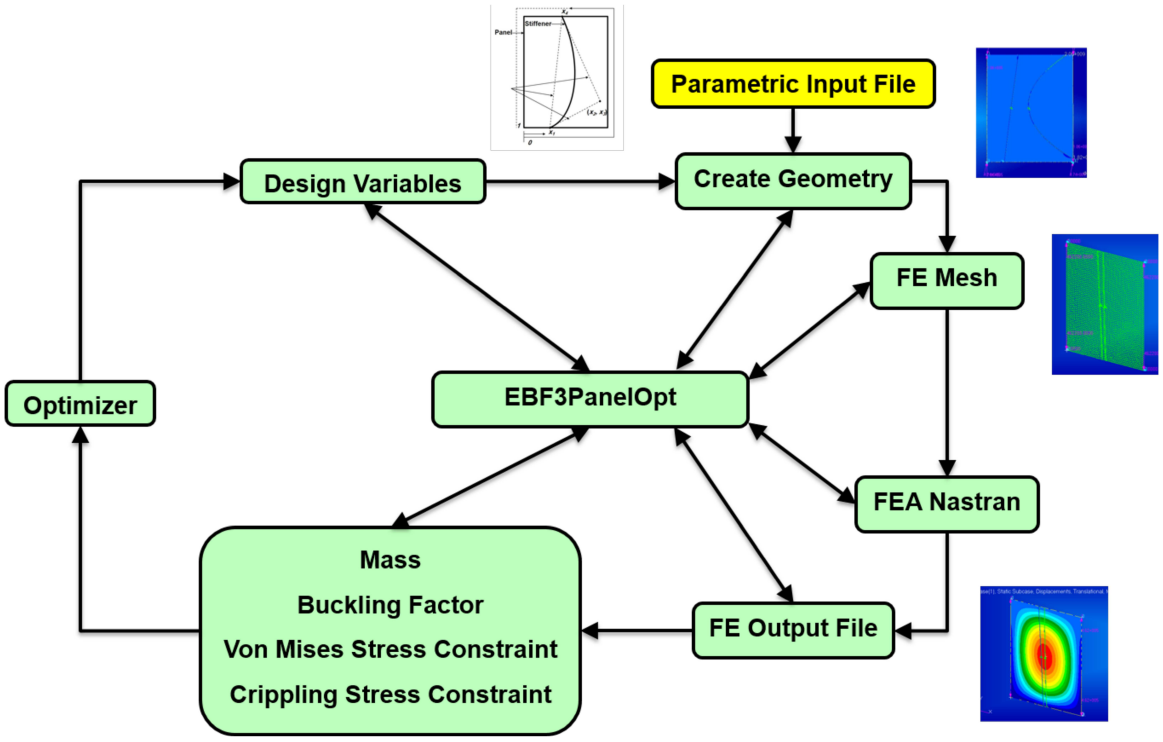
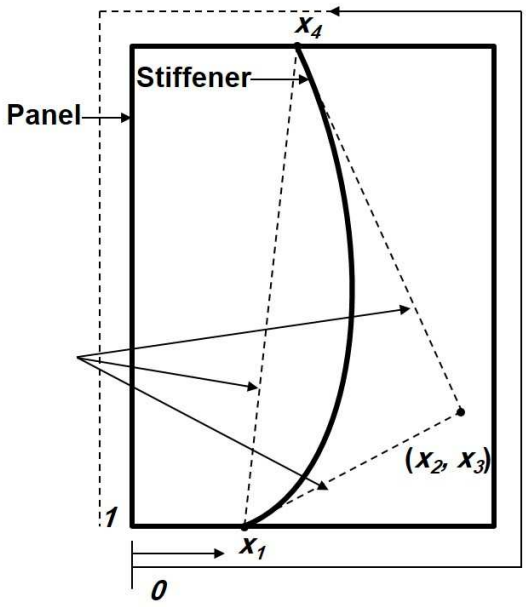


Figure 6. Flowchart depicting the EBF3PanelOpt framework.

Mises stress factor, and crippling factor, which finally becomes the input to an optimizer. More details about the framework EBF3PanelOpt can be found in [40].

A key feature of EBF3PanelOpt is the ability to specify the geometry of the stiffened panel parametrically. The (parametric) design variables used in this framework determine the stiffeners' shape, position, height, and thickness, and the panel thickness. An example of how the stiffener geometry is defined in terms of geometric parameter lower and upper bounds is shown in Figure 7 (right). Four design variables are used to calculate the shape and position of every stiffener, as shown in Figure 7 (left). The stiffener's curve is represented using a third order uniform rational B-spline using two end points (defined by x_1 and x_4) and a control point (x_2, x_3), which guarantees that the stiffener always remains in the panel area. The two stiffener end points lie on the panel's perimeter and hence can each be represented with a single value (x_1 and x_4) that always lies between zero and one. A single perimeter curve is created in Patran and used to localize the end points of the stiffeners using parametric extraction. The values of x_2 and x_3, also between zero and one, determine the control point on the panel surface. Thus a panel



	Min	Max	
x_1	0.0	0.23	Stiffener I Geometry
x_2	0.0	1.0	
x_3	0.0	1.0	
x_4	0.5	0.73	
x_5	0.0	0.23	Stiffener II Geometry
x_6	0.0	1.0	
x_7	0.0	1.0	
x_8	0.5	0.73	
x_9	0.01	0.05	Stiffeners Heights
x_{10}	0.01	0.05	
x_{11}	0.001	0.008	Stiffeners Thickness
x_{12}	0.001	0.008	
x_{13}	0.001	0.004	Panel Thickness

Figure 7. Design variables of the stiffener (left).
Definition and range of the design variables for the optimization process (right).

with nearly arbitrary geometry, along with the stiffeners, can be represented using this parametrization.

4.2. Implementation of EBF3PanelOpt as a Service

In order to achieve truly distributed objective function evaluations, the optimization framework EBF3PanelOpt is implemented as an analysis provider. Analysis providers can be dynamically distributed over a variety of computational resources with the help of SORCER's JavaSpaces technology, which implements a loosely coupled distributed computing system across the network. JavaSpaces not only enables computers on the network to communicate reliably, but also provides load balancing capability to cope with dynamic resources. Hence, computing resources can be added during the course of an optimization study, thereby enhancing productivity. The JavaSpaces technology provides a type of shared memory where exertion evaluators can drop tasks they wish to be processed by service providers. When services providers are started on the network, they use Jini discovery mechanisms to find *spaces* on the network. If unprocessed tasks reside in the space, the provider picks up the task from the space and executes the appropriate service. Once execution is completed,

the task is returned to the space and marked as processed. Processed tasks are removed from the space by the evaluator.

For the parallel implementation of QNSTOP that constructs a table of runs (a modification of the OpenMP parallel code QNSTOPP), the EBF3PanelOpt provider is 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. The providers are started on multiple machines to distribute the work. During optimization, the model provider first creates child instances for every row in the table and drops these tasks into the space. Then, based on the number of worker threads, each EBF3PanelOpt provider picks up unprocessed tasks from the space, executes these tasks in parallel, and upon completion, returns them to the space.

Figure 8 (top) shows the use of JavaSpaces technology in a dynamic distributed computing environment for the panel optimization studies. An alternative to JavaSpaces is Catalog, referred to here as SORCER/Catalog and shown in Figure 8 (bottom), that has advantages in certain contexts, such as multicore or single large parallel distributed memory machines. Here, service providers publish proxies to the catalog. The requestor passes a service request to the catalog, which "matches" the service request with one of the proxies. If there are multiple proxies that match the request, the catalog uses an algorithm such as round robin to select the proxy. This proxy is then passed to the requestor, who uses the proxy to make the remote call directly to the provider (as in Figure 1). In Figure 8 (bottom) the EBF3PanelOpt providers #1 and #2 publish their proxies with the catalog (the QNSTOPP provider and the EBF3PanelOpt Model provider also publish proxies, but this is not shown in the figure). The JNI wrapper submits a table to the EBF3PanelOpt Model provider. The EB3PanelOpt Model provider submits a service request to the catalog, which satisfies that request and passes the proxies for the EBF3PanelOpt providers to the model, which then makes a call on the proxies to access the EBF3PanelOpt providers.

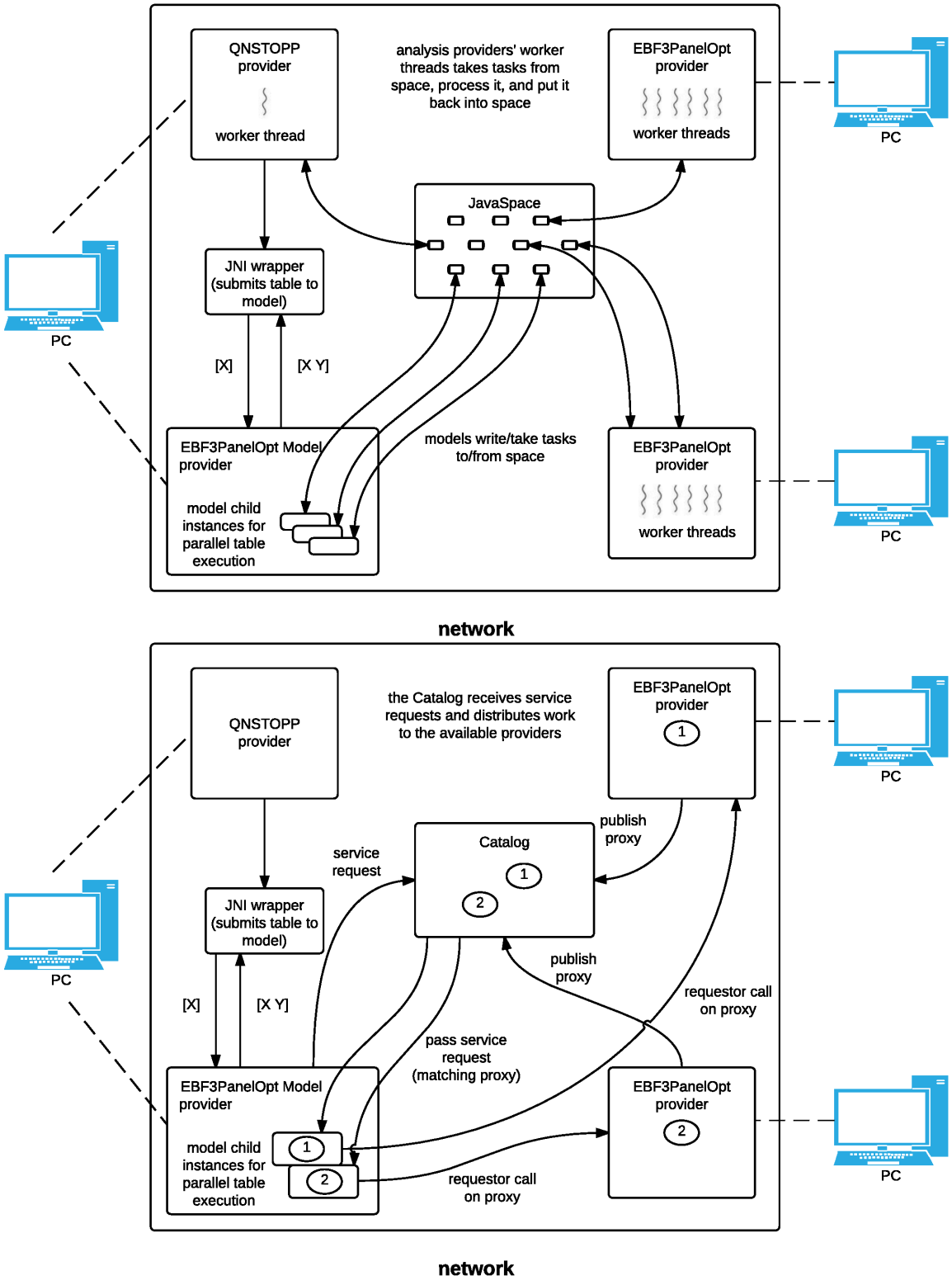


Figure 8. Use of JavaSpaces technology (top) and Catalog technology (bottom) in a dynamic distributed computing environment for the panel optimization studies.

Chapter 5.

NUMERICAL RESULTS

This chapter presents results for optimization of curvilinear blade-stiffened panels using VTDIRECT95 and QNSTOP. The chapter is further divided into two subchapters — Experiment 1 and Experiment 2. Experiment 1 presents optimization results for a stiffened panel of dimensions $0.6096m \times 0.7112m$; Experiment 2 presents optimization results for a stiffened panel of dimensions $0.4064m \times 0.5080m$.

All experiments presented here are conducted on Intel machines each with 16GB of memory and a single quad-core processor, in which each core supports hyperthreading. The experiments are conducted using GNU Fortran 4.9.1, GNU C 4.9.1, Python 2.6.6, Open MPI 1.8.1, and Java 1.8.0_25 on x86_64 RHEL running CentOS 6.6. The framework EBF3PanelOpt is configured to use Nastran 2014 and Patran 2014. For all experiments, the mesh size parameter in MD-Patran is set to 0.01 during mesh generation for the stiffened panels. This implies that the average elemental edge length would be approximately 0.01.

5.1. Experiment 1

The framework EBF3PanelOpt is applied to a panel with four curvilinear stiffeners and subjected to biaxial normal and shear loads (Figure 9). The panel is fixed such that rotations and transverse displacements are not allowed, but in-plane displacements are allowed. The panel dimensions are $0.6096m \times 0.7112m$, and the applied loads are $N_{XX} = 15,761\,N/m$, $N_{YY} = 140,280\,N/m$, $N_{XY} = 44,307\,N/m$. In [40], the framework EBF3PanelOpt was used with the heuristic particle swarm optimization technique followed by the gradient based method of feasible directions to approximate a locally optimal design for this same flat rectangular panel. The material properties, load cases, sizing design variables' constraints, and other details can be found in [40].

Using the inputs given in [40], optimization is carried out with pVTdirect. For optimization with pVTdirect, a feasible box inside the sizing design variables' constraints was chosen. The values (mass, buckling factor, Kreisselmeier-Steinhauser criterion, and

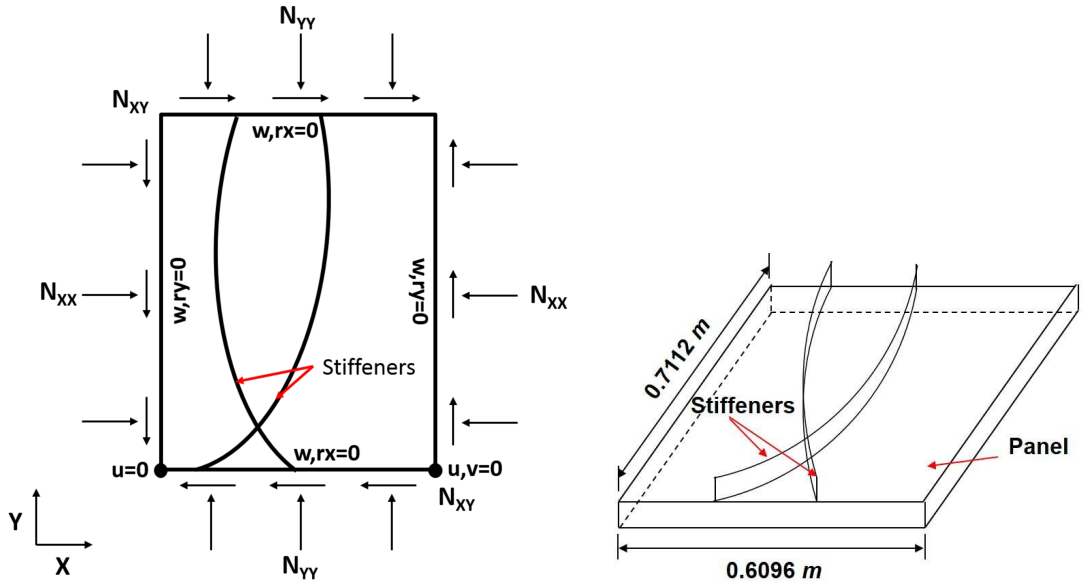


Figure 9. Curvilinear stiffened panels under combined shear and normal loads.

Table 1

Pylon wing panel, $0.6096m \times 0.7112m$*, optimization results, four stiffeners.*

	PSO + GBO [40]	pVTdirect
Mass (kg)	3.2337	3.1269
Buckling factor	1.0015	0.9890
KSC	0.2109	0.2074
Crippling Criterion	0.9137	0.8290
Stiffener 1 height (m)	4.8865E-02	4.1333E-02
Stiffener 2 height (m)	4.8893E-02	4.1333E-02
Stiffener 3 height (m)	1.3403E-02	1.1666E-02
Stiffener 4 height (m)	2.9993E-02	2.9000E-02
Stiffener 1 thickness (m)	1.0003E-03	1.1666E-03
Stiffener 2 thickness (m)	1.1290E-03	1.1000E-03
Stiffener 3 thickness (m)	2.5810E-03	1.8333E-03
Stiffener 4 thickness (m)	1.0002E-03	1.1666E-03
Plate thickness (m)	2.4502E-03	2.4000E-03

crippling criterion) and designs reported in [40] were approximately reproduced (cf. Table 1). However, these results and the algorithmic efficiency cannot be compared to those in [40], because [40] did not report the number of function evaluations used by their heuristic algorithm.

The von Mises stress distribution and displacement for results obtained by pVTdirect are shown in Figure 10.

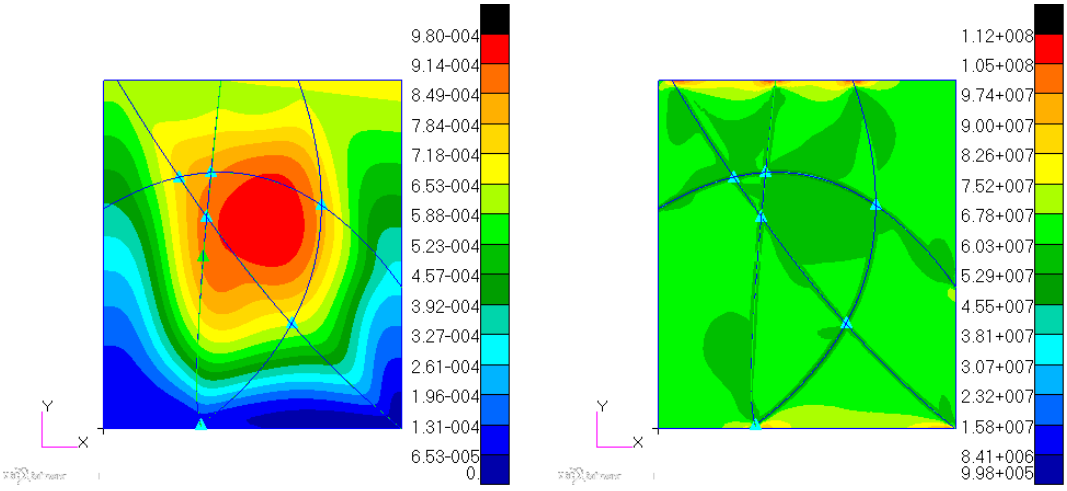


Figure 10. Pylon wing panel, $0.6096m \times 0.7112m$, four stiffeners, optimization using pVTdirect: (a) displacement (m), and (b) von Mises stress distribution (Pa).

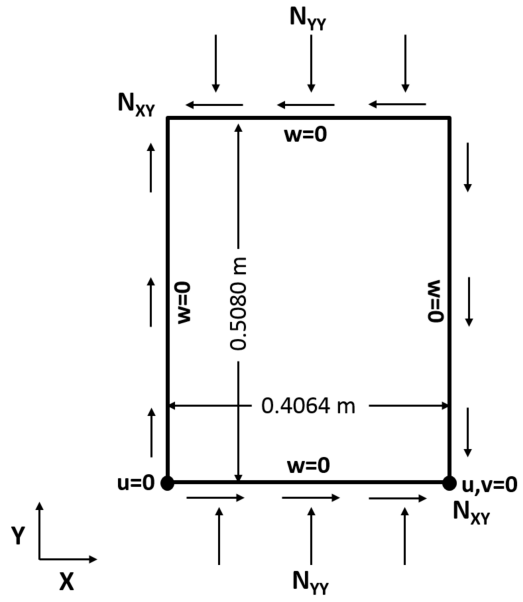


Figure 11. Conventional aircraft wing panel geometry along with the loads.

5.2. Experiment 2

Figure 11 shows a simply supported flat rectangular $0.4064m \times 0.5080m$ panel representing a large wing engine pylon rib, which is optimized for minimum mass using VTDIRECT95 and QNSTOP. The material used for these panels is the aluminum alloy *Al 2139*, whose properties are given in Table 2. The panel is subjected to combined compression and shear

Table 2

Material properties for Al 2139.

Young's modulus (GPa)	73.085
Poisson's ratio	0.33
Yield Stress (MPa)	427.47
Density (kg/m^3)	2795.00

in-plane loads ($N_{YY} = 462,200\,N/m$ and $N_{XY} = 106,100\,N/m$). The problem of a simply supported panel with two curvilinear stiffeners, subjected to the same combined compression and shear in-plane loads, and a crack with nonprescribed location is examined in [26].

The results for the optimization of curvilinear blade-stiffened panels subjected to the combined compression and shear in-plane loads mentioned above, containing two stiffeners (Case 1) and four stiffeners (Case 2), using the subroutines VTdirect, pVTdirect, QNSTOPS, and QNSTOPP are presented below.

5.2.1. Case 1: Optimization of curvilinear blade-stiffened panels containing two stiffeners. The (parametric) design variables that represent the geometry of the panel, stiffeners' position, height, and thickness, and the panel thickness are given in Table 3. For optimization with VTdirect, the design variables' constraints in Table 3 are the lower and upper bounds on X. The stopping condition is a limit of 1000 on the number of objective function evaluations. Additionally, for pVTdirect, the number of processes is set to four. Since pVTdirect is incompatible with SORCER/JavaSpace/table model query, results for this case are omitted.

For optimization with (the serial subroutine) QNSTOPS, the sizing design variables' constraints in Table 3 are the lower and upper bounds on the design vector X, which is an ellipsoid center in the QNSTOP algorithm. The initial ellipsoid center X is the mean of the lower and upper bounds. It is generally desirable to run QNSTOP from multiple start points [4]. The optional argument NSTART, set to 5 here, is the number of start points to be automatically generated when the optional argument SWITCH = 3. These start points, of which the initial input X is the first, are derived by Latin hypercube sampling in the feasible box. The size N of the experimental design used in each quasi-Newton iteration is set to 20. QNSTOPS is run in mode 'G' (deterministic mode) and the factor by which TAU (the initial radius for the ellipsoidal design region) is decayed is specified by the optional

Table 3

The sizing design variables' constraints.

	Lower bound (m)	Upper bound (m)
Stiffeners 1 & 2 Geometry (x_1, x_5)	0.0	0.23
Stiffeners 1 & 2 Geometry (x_2, x_6)	0.0	1.0
Stiffeners 1 & 2 Geometry (x_3, x_7)	0.0	1.0
Stiffeners 1 & 2 Geometry (x_4, x_8)	0.5	0.73
Stiffener Height	0.01	0.05
Stiffener Thickness	0.001	0.007
Plate Thickness	0.001	0.007

argument GAIN, which is set to 1.0. The stopping rule is a limit of 1000 on the number of objective function evaluations (200 evaluations per start point).

Additionally, for (the parallel subroutine) QNSTOPP, the optional argument OMP is set to 2, which corresponds to parallelizing just the loop over the sampling point objective function evaluations. The environment variable OMP_NUM_THREADS is set to 4 to match the hardware capability. These input values apply to the JNI runs without SORCER; recall that QNSTOPP has to be modified for SORCER as described in Chapter 3.5.2. QNSTOPP for SORCER does not use OpenMP to achieve parallelism, but rather instead interacts with the model provider via a table model query, which in turn spawns a child instance for each row in the table for parallel execution of the table row evaluations. Hence, for all QNSTOPP runs with SORCER, OMP should always be set to 0 (no OpenMP parallelization) and the environment variable OMP_NUM_THREADS should be set to 1. It is also important that the number of rows in the table be compatible with the machine's underlying hardware. For all QNSTOPP runs with SORCER presented in this thesis, two identical machines are used. The program carrying out optimization as a service and the model provider are started on one machine, and the EBF3PanelOpt provider on the other. For all QNSTOPP runs, the table is configured with four rows, resulting in four concurrent objective function evaluations.

The results for the two-stiffener panel optimization using the subroutines VTdirect, pVTdirect, QNSTOPS, and QNSTOPP are shown in Table 4. The execution times for pVTdirect, VTdirect, QNSTOPS, and QNSTOPP, with and without SORCER, are listed

Table 4

Pylon wing panel, $0.4064m \times 0.5080m$, optimization results, two stiffeners.

	VTdirect	QNSTOPS	QNSTOPP
Mass (kg)	2.5274	2.5009	2.4189
Buckling factor	0.9825	0.9506	0.9879
KSC	0.2956	0.3253	0.3233
Crippling Criterion	0.7206	0.3489	0.3212
No. of function evaluations	1131	950	950
Stiffener 1 height (m)	3.0000E-02	3.5613E-02	2.2977E-02
Stiffener 2 height (m)	2.5555E-02	3.0199E-02	3.1799E-02
Stiffener 1 thickness (m)	4.0000E-03	4.3402E-03	3.5181E-03
Stiffener 2 thickness (m)	1.3333E-03	3.6315E-03	3.8770E-03
Plate thickness (m)	4.0000E-03	3.6784E-03	3.6860E-03

Table 5

Execution time (s) for pylon wing panel optimization (two stiffeners).

	VTdirect	pVTdirect	QNSTOPS	QNSTOPP	E_p
With SORCER and script robustness	13,009	N/A	11,388	3,545	0.80
With SORCER, without script robustness	8,957	N/A	7,994	2,542	0.79
With SORCER/Catalog, without script robustness	8,487	N/A	7,597	2,458	0.77
Without SORCER and script robustness	8,460	2,924	7,560	2,309	0.82

in Table 5. The last column in the table represents the parallel efficiency with QNSTOPP. For the runs that do not involve SORCER, the parallel efficiency is

$$E_p = \frac{\big((\text{serial QNSTOPS time})/(\text{parallel QNSTOPP time})\big)}{(\text{total number of OMP threads})}.$$

For the runs of (serial) QNSTOPS with SORCER and (modified parallel) QNSTOPP with SORCER, the parallel efficiency is

$$E_p = \frac{\big((\text{QNSTOPS time})/(\text{modified QNSTOPP time})\big)}{(\text{number of function evaluation threads})}.$$

In this experiment, for QNSTOPP runs without SORCER, OMP_NUM_THREADS = 4, and for QNSTOPP runs with SORCER, the number of rows in the model query table is four. In Tables 5, 8, and 9 "script robustness" refers to a Java utility `GenericUtil` separate

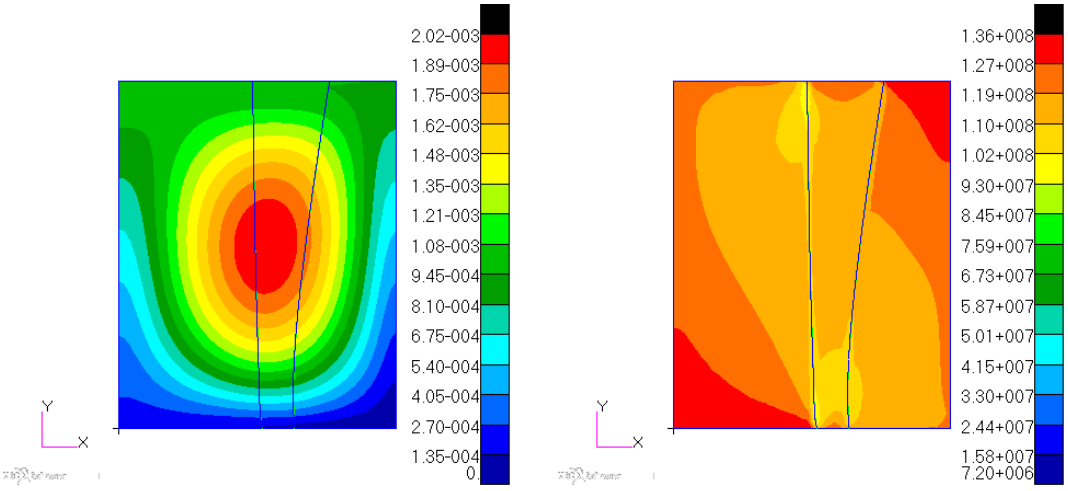


Figure 12. Pylon wing panel, $0.4064m \times 0.5080m$, two stiffeners, optimization using VTdirect: (a) displacement (m), and (b) von Mises stress distribution (Pa).

from SORCER, recommended for use of scripts in production distributed computing, that increases the robustness of scripts and communication links across different operating systems. "SORCER/Catalog" is an alternative to SORCER with JavaSpaces that is more appropriate for a multicore machine. The Catalog concept from a precursor of SORCER is essentially the service registry shown in Figure 1. The parallel efficiencies E_p for QNSTOPP runs without and with SORCER directly measure the overhead cost incurred with SORCER: using SORCER without script robustness rather than the lower level OpenMP/MPI parallel directives reduces the parallel efficiency slightly. Using script robustness with SORCER (most general and reliable production mode) for all the codes — VTdirect, QNSTOPS, QNSTOPP — increases the execution time significantly but changes the parallel efficiency only slightly. Note that the cost of script robustness and reliable distributed execution is roughly independent of the problem dimension and the cost of the function evaluations, so with more expensive function evaluations the parallel efficiency would improve.

The von Mises stress distribution and displacement for results obtained by VTdirect, QNSTOPS, and QNSTOPP are shown in Figures 12, 13, and 14, respectively. For all three cases, the buckling constraint value is close to 1, indicating that the corresponding masses are near optimal. The KSC criterion value is slightly higher for the QNSTOPS and QNSTOPP results indicating that the panel is more stressed for better designs. The crippling criterion value for the best mass obtained with VTdirect is moderately close to saturation compared to those obtained with QNSTOPS and QNSTOPP. This indicates that the stiffener is more prone to crippling when one or more flanges buckle in a local buckling mode with wavelength unrelated to the length of the beam.

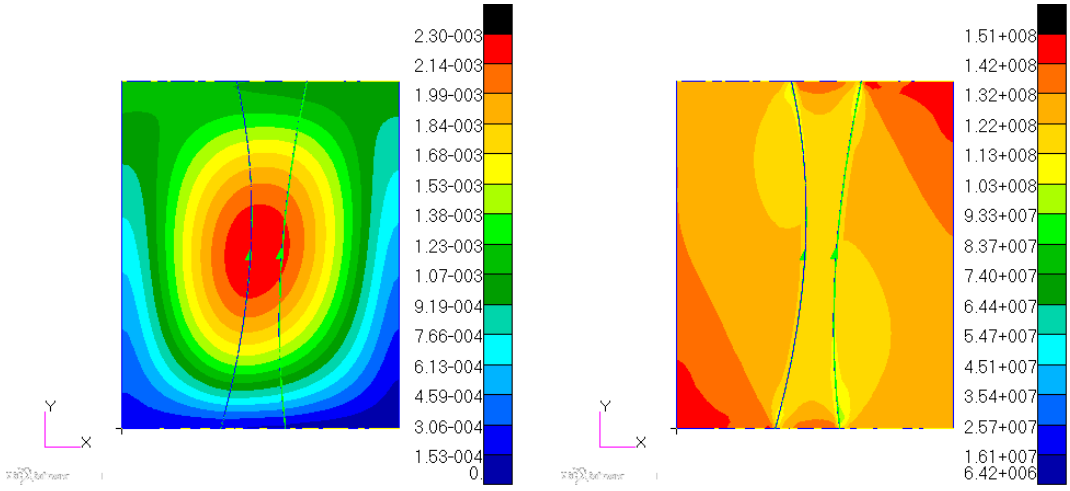


Figure 13. Pylon wing panel, $0.4064m \times 0.5080m$, two stiffeners, optimization using QNSTOPS: (a) displacement (m), and (b) von Mises stress distribution (Pa).

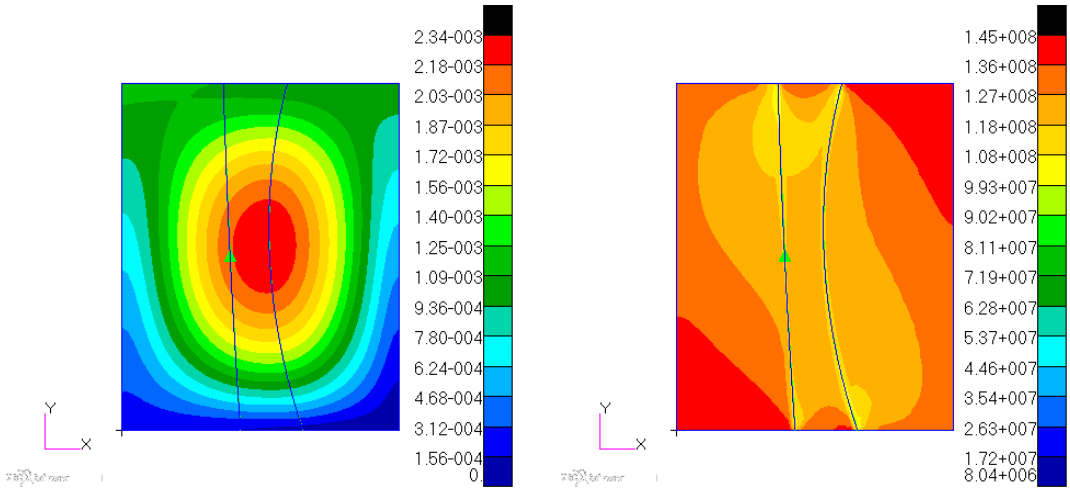


Figure 14. Pylon wing panel, $0.4064m \times 0.5080m$, two stiffeners, optimization using QNSTOPP: (a) displacement (m), and (b) von Mises stress distribution (Pa).

5.2.2. Case 2: Optimization of curvilinear blade-stiffened panels containing four stiffeners. The design variables that represent the geometry of the panel, stiffeners' position, height, and thickness, and the panel thickness are given in Table 6. For optimization with VTdirect, the design variables' constraints in Table 6 are the lower and upper bounds on X. Similar to Case 1, the stopping condition is a limit of 1000 on the number of objective function evaluations. For pVTdirect, the number of processes is set to 4. Since pVTdirect is incompatible with SORCER/JavaSpace/table model query, results for this case are omitted.

Table 6

The sizing design variables' constraints.

	Lower bound (m)	Upper bound (m)
Stiffeners 1 & 2 Geometry (x_1, x_5)	0.0	0.23
Stiffeners 1 & 2 Geometry (x_2, x_6)	0.0	1.0
Stiffeners 1 & 2 Geometry (x_3, x_7)	0.0	1.0
Stiffeners 1 & 2 Geometry (x_4, x_8)	0.5	0.73
Stiffener 3 Geometry (x_9)	0.23	0.5
Stiffener 3 Geometry (x_{10})	0.0	1.0
Stiffener 3 Geometry (x_{11})	0.0	1.0
Stiffener 3 Geometry (x_{12})	0.73	1.0
Stiffener 4 Geometry (x_{13})	0.15	0.5
Stiffener 4 Geometry (x_{14})	0.0	1.0
Stiffener 4 Geometry (x_{15})	0.0	1.0
Stiffener 4 Geometry (x_{16})	0.5	1.0
Stiffener Height	0.01	0.05
Stiffener Thickness	0.001	0.007
Plate Thickness	0.001	0.007

For optimization with (the serial subroutine) QNSTOPS, the sizing design variables' constraints in Table 6 are the lower and upper bounds on the design vector X, which is an ellipsoid center in the QNSTOP algorithm. The initial ellipsoid center X is the mean of the lower and upper bounds. As in the previous case, the optional argument NSTART (number of start points) is set to 5 when the optional argument SWITCH = 3. The size N of the experimental design used in each quasi-Newton iteration is set to 40. QNSTOPS is run in mode 'G' (deterministic mode) and the factor by which TAU (the initial real radius for the ellipsoidal design region) is decayed is specified by the optional argument GAIN, which is set to 1.0. The stopping rule is a limit of 1000 on the number of objective function evaluations (200 evaluations per start point).

Additionally, for (the parallel subroutine) QNSTOPP, the optional argument OMP is set to 2. The environment variable OMP_NUM_THREADS is set to 4 to match the hardware capability. For all QNSTOPP runs with SORCER, OMP is set to 0 and the environment variable OMP_NUM_THREADS is set to 1. The table that is passed from the JNI wrapper to the model provider is configured to have four rows, resulting in four concurrent objective function evaluations.

The results for the four-stiffener panel optimization using the subroutines VTdirect, QNSTOPS, and QNSTOPP are shown in Table 7. The execution times for pVTdirect,

Table 7

Pylon wing panel, $0.4064m \times 0.5080m$*, optimization results, four stiffeners.*

	VTdirect	QNSTOPS	QNSTOPP
Mass (kg)	2.5472	2.7024	2.8863
Buckling factor	0.9901	0.9667	0.91172
KSC	0.2901	0.3277	0.3378
Crippling Criterion	0.5364	0.3676	0.2567
No. of function evaluations	1165	825	825
Stiffener 1 height (m)	3.0000E-02	2.8342E-02	3.2533E-02
Stiffener 2 height (m)	1.6666E-02	2.4909E-02	2.5158E-02
Stiffener 3 height (m)	3.0000E-02	3.3505E-02	2.9751E-02
Stiffener 4 height (m)	1.6666E-02	2.9939E-02	3.0364E-02
Stiffener 1 thickness (m)	2.0000E-03	3.9661E-03	4.6432E-03
Stiffener 2 thickness (m)	2.0000E-03	3.1711E-03	4.3679E-03
Stiffener 3 thickness (m)	2.0000E-03	3.7376E-03	4.0771E-03
Stiffener 4 thickness (m)	2.0000E-03	3.8998E-03	3.5480E-03
Plate thickness (m)	4.0000E-03	3.6572E-03	3.8516E-03

VTdirect, QNSTOPS, and QNSTOPP, with and without SORCER and/or script robustness (described earlier), are listed in Table 8. The last column in the table represents the parallel efficiency with QNSTOPP.

In this experiment, note that the numbers of function evaluations (in Table 7) for VTdirect and QNSTOP* are different than those for Case 1 (in Table 4), since these numbers depend on the problem dimension and the sample size (per start point for QNSTOP*). The trends and implications of Table 8 are similar to those of Table 5. Even though the problem size doubled (from 13 to 25), the parallel efficiencies decreased because the number of function evaluations by QNSTOPP was less for Case 2 than for Case 1.

The von Mises stress distribution and displacement for results obtained by VTdirect, QNSTOPS, and QNSTOPP are shown in Figures 15, 16, and 17, respectively. For all three cases, the buckling constraint is close to 1, indicating that the corresponding masses are nearly optimal. The design of Figure 17 is interesting from a stability point of view, since the two close stiffeners in the middle of the panel behave like supports to clamp the panel in the middle and eliminate, therefore, the low order buckling modes. Moreover, the other two stiffeners divide the panel into smaller subpanels and help increase its buckling load.

The computational expense of each objective function evaluation is listed in Table 9, where n is the problem dimension. To estimate the average computational expense for an objective function evaluation, runs were made with VTdirect for a total of 100 function

Table 8

Execution time (s) for pylon wing panel optimization (four stiffeners).

	VTdirect	pVTdirect	QNSTOPS	QNSTOPP	E_p
With SORCER and script robustness	14,450	N/A	10,370	3,676	0.71
With SORCER, without script robustness	10,384	N/A	7,451	2,697	0.69
With SORCER/Catalog, without script robustness	9,815	N/A	7,088	2,615	0.68
Without SORCER and script robustness	9,786	3,789	7,052	2,408	0.73

Table 9

Objective function evaluation time (s) for pylon wing panel (two and four stiffeners).

	$n = 13$	$n = 25$
With SORCER and script robustness	11.13	12.90
With SORCER, without script robustness	7.36	9.14
Without SORCER and script robustness	7.32	9.10

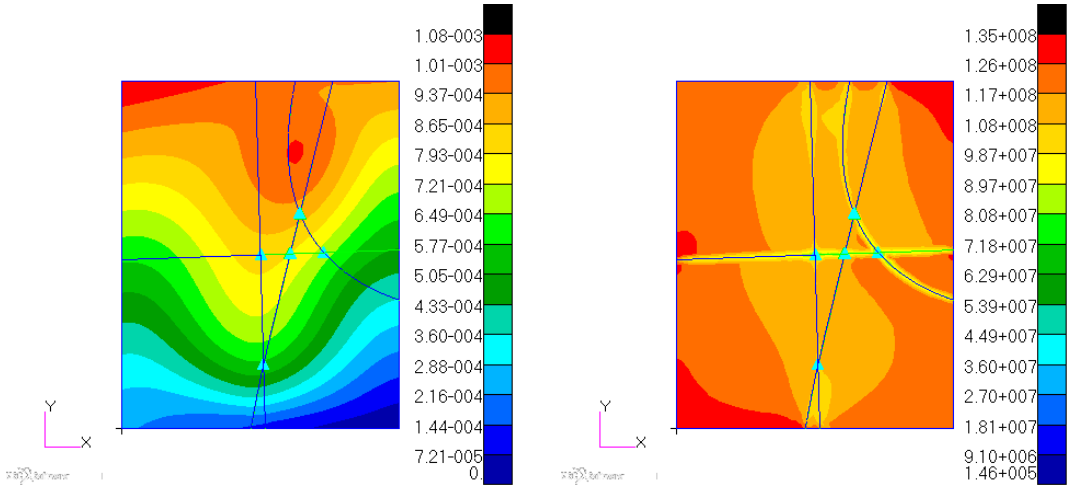


Figure 15. Pylon wing panel, $0.4064m \times 0.5080m$, four stiffeners, optimization using VTdirect: (a) displacement (m), and (b) von Mises stress distribution (Pa).

evaluations, recording the time between when the arguments are passed to the evaluation code, and the time when the function value comes back. With SORCER with and without script robustness, runs were made with VTdirect for a total of 100 function evaluations,

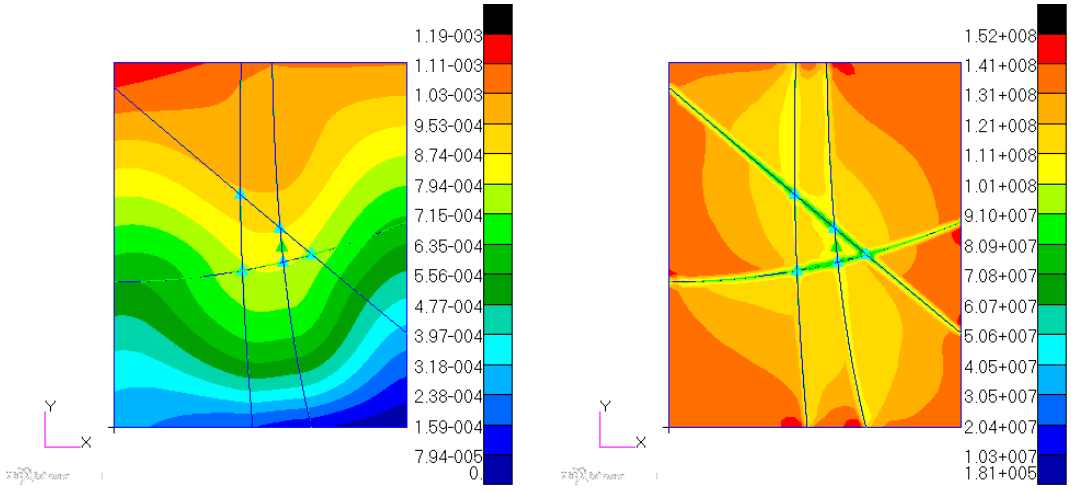


Figure 16. Pylon wing panel, $0.4064m \times 0.5080m$, four stiffeners, optimization using QNSTOPS: (a) displacement (m), and (b) von Mises stress distribution (Pa).

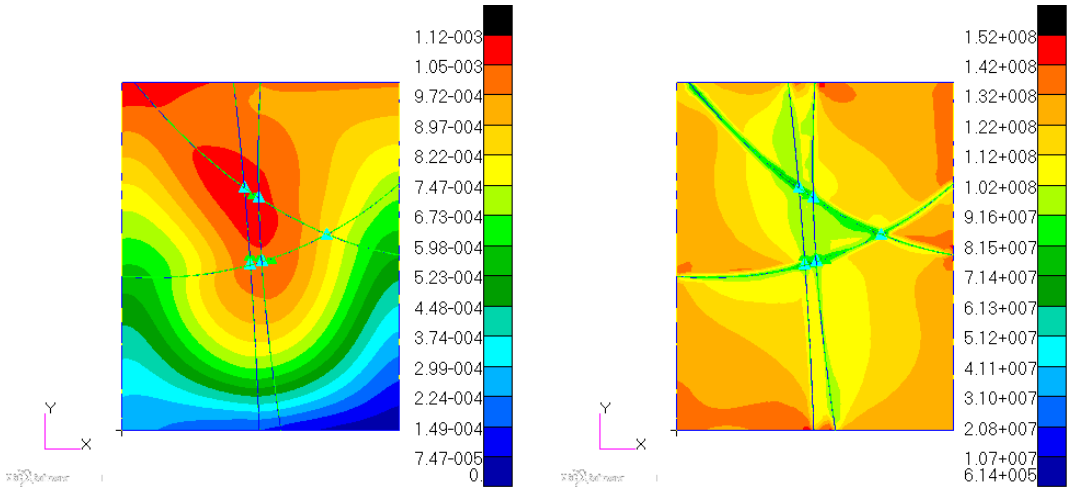


Figure 17. Pylon wing panel, $0.4064m \times 0.5080m$, four stiffeners, optimization using QNSTOPP: (a) displacement (m), and (b) von Mises stress distribution (Pa).

recording the time between when the call to *execute* on the analysis provider that provides EBF3PanelOpt as a service is made and when the function value is sent to the model client. The SORCER with script robustness overhead per function evaluation ($\approx 4\,s$) is about the same for both problem sizes, and without script robustness the SORCER overhead is negligible. Robustness and portability do not come cheap.

Chapter 6.

CONCLUSION AND FUTURE WORK

From Table 9, the SORCER with script robustness overhead per function evaluation is about four seconds (essentially all due to script and communication link robustness across different operating systems), and similar overhead would be expected had the VTdirect, QNSTOPS, and QNSTOPP runs without SORCER been done with script robustness. With 1000 function evaluations on the order of 10 seconds each, this SORCER with script robustness overhead is significant as reflected in Table 8. The parallel efficiency E_p is a fair measure of the extent to which parallel hardware resources are being utilized efficiently, and $E_p > 0.5$ is considered acceptable. If 1000 function evaluations taking one hour each were done instead, the SORCER with script robustness overhead would be relatively negligible and $E_p \approx 1.0$. Apart from the script robustness overhead for function evaluations, Tables 5 and 8 show that the other SORCER overhead is not significant for expensive function evaluations in a distributed computing environment. MDO may involve function evaluations ranging in cost over several orders of magnitude ($< 10\,s$ to $> 1000\,s$), so the conclusion is that the right SORCER paradigms (JavaSpaces, though the most general approach, is but one of several approaches) must be used in the right contexts, and no single SORCER paradigm is a general purpose solution to parallel and distributed computing for MDO. On the continuum of distributed computing technology (MPI, Globus, Legion, Condor, SORCER), SORCER is at the heavyweight end.

The use of JNI to wrap the corresponding native code provided a clean and elegant interface between the optimization algorithm and the Java block that evaluates the objective for a design point. While JNI is associated with high development overhead, JNA (Java Native Access) is a library that provides easy access to native shared libraries without boiler plate/glue code. However, JNA has more execution overhead and is slower than JNI. Some of the disadvantages of JNI include: need for boiler plate code in C, poor performance, and weak security. While JNI is the current standard programming interface for interfacing native methods with Java, the Java FFI (Foreign Function Implementation)

has been proposed to address the difficulties of JNI. The Java FFI is a metadata system that will allow access to native functions with native memory management at the Java level. This feature would not require the developer to deploy boiler plate code or have expertise in JVM internals. As Java FFI matures, it would be interesting to replace the existing JNI code with Java FFI and carry out trade-off studies between performance and the cost of development.

Two different optimization algorithms widely used in multidisciplinary engineering design — VTDIRECT95 and QNSTOP — were implemented as services on a SORCER grid. The EBF3PanelOpt framework was successfully integrated with SORCER, hence facilitating the optimization of curvilinearly stiffened panels in a truly distributed manner. Further, the implementation of the EBF3PanelOpt framework as an analysis provider within SORCER provided flexibility; the objective function evaluations could be moved to less-burdened machines as and when required. With the JavaSpaces technology, computers could be added on the fly, providing flexibility, and enabling distributed parallelism and load balancing across computational resources in a dynamically scalable environment.

It should also be mentioned that the installation of SORCER is far from routine—SORCER is currently a research code with limited documentation and requires considerable knowledge of network computing to install and utilize. Future work includes modifying the parallel subroutine pVTdirect of VTDIRECT95 to accommodate chunking of function evaluations at a synchronization point, which would then be compatible with a master-slave paradigm in which design points are readily accessible by a master. This implementation could be integrated with the SORCER/JavaSpace/table model query paradigm, thus providing both VTDIRECT95 and QNSTOP as robust distributed SORCER services.

REFERENCES

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

[2] Andrew T. M., Amos B. D., Easterling D. R., Oguz C., Baumann W. T., Tyson T. T., Watson L. T., "Global Parameter Estimation for a Eukaryotic Cell Cycle Model in Systems Biology", in *Summer Simulation Multi-Conference*, Monterey, CA, July 2014.

[3] Amos B. D., Easterling D. R., Watson L. T., Castle B. S., Trosset M. W., and Thacker W. I., "Fortran 95 Implementation of QNSTOP for Global and Stochastic Optimization", in *22nd High Performance Computing Symposium (HPC 2014)*, Tampa, Florida, April 2014.

[4] Amos B. D., Easterling D. R., Watson L. T., Thacker W. I., Castle B. S., and Trosset M. W., "Algorithm XXX: QNSTOP: Quasi-Newton Algorithm for Stochastic Optimization", Technical Report, 2014-07, Virginia Polytechnic Institute and State University, Blacksburg, VA, 2014.

[5] Burton S. A., Alyanak E. J., and Kolonay R. M., "Efficient Supersonic Air Vehicle Analysis and Optimization Implementation using SORCER", in *AIAA 2012-5520, 12th AIAA Aviation Technology, Integration, and Operations (ATIO) Conference and 14th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Indianapolis, Indiana, Sept. 2012.

[6] Baker C. A., Grossman B., Haftka R. T., Mason W. H., and Watson L. T., "HSCT Configuration Design Space Exploration using Aerodynamic Response Surface Approximations", in *AIAA 98-4803, 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Saint Louis, MO, Sept. 1998.

[7] Bartholomew-Biggs M. C., Parkhurst S. C., and Wilson S. P., "Global Optimization Approaches to an Aircraft Routing Problem", *European Journal of Operational Research*, Vol 146(2), pp. 417–431, April 2003.

[8] Baker C. A., Watson L. T., Grossman B., Haftka R. T., and Mason W. H., "Study of a Global Design Space Exploration Method for Aerospace Vehicles", in *AIAA 2000-4763, 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Long Beach, CA, Sept. 2000.

[9] Baker C. A., Watson L. T., Grossman B., Mason W. H., and Haftka R. T., "Parallel Global Aircraft Configuration Design Space Exploration", Technical Report MAD 2006-06-28, Virginia Polytechnic Institute and State University, Blacksburg, VA, 2000.

[10] Boisvert R. F., Moreira J., Philippsen M., and Pozo R., "Java and Numerical Computing", *IEEE Computing in Science and Engineering*, Vol 3(2), pp. 18–24, March/April 2001.

[11] Castle B. S., *Quasi-Newton Methods for Stochastic Optimization and Proximity-based Methods for Disparate Information Fusion*, Ph.D. thesis, Indiana University, Bloomington, IN, 2012.

[12] Carter R. G., Gablonsky J. M. Patrick A., Kelly C. T., and Eslinger O. J., "Algorithms for Noisy Problems in Gas Transmission Pipeline Optimization", *Optimization and Engineering*, Vol. 2(2), pp. 139–157, June 2001.

[13] Easterling D. R., Watson L. T., Madigan M. L., Castle B. S., and Trosset M. W., "Direct Search and Stochastic Optimization applied to Two Nonconvex Nonsmooth Problems", in *20th High Performance Computing Symposium*, Orlando, FL, March 2012.

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

[15] Foster I., and Kesselman C., "Globus: A Metacomputing Infrastructure Toolkit", *International Journal of Supercomputer Applications*, Vol. 11(2), pp. 115–128, June 1997.

[16] Ghommem M., Hajj M. R., Stanford B. K., Watson L. T., and Beran P. S., "Global and Local Optimization of Flapping Kinematics", in *AIAA 2012-1983, 53rd AIAA/ASME/ASCE/AHS/-ASC Structures, Structural Dynamics, and Materials Conference*, Honolulu, Hawaii, April 2012.

[17] Georgakopoulous D. and Papazoglou M. P., *Service-Oriented Computing*, The MIT Press, Cambridge, 2008.

[18] Goel S., Talya S., and Sobolewski M., "Preliminary Design Using Distributed Service-based Computing", in *12th ISPE International Conference on Concurrent Engineering*, Fort Worth, Texas, July 2005.

[19] Grimshaw A. S., and Wulf W. A., "The Legion Vision of a Worldwide Virtual Computer", *Communications of the ACM*, Vol. 40(1), pp. 39–45, Jan. 1997.

[20] Gao D. Y., Watson L. T., and Easterling D. R., "Solving the Canonical Dual of Box- and Integer-constrained Nonconvex Quadratic Programs via a Deterministic Direct Search Algorithm", *Optimization Methods and Software*, Vol. 28(2), pp. 313–326, April 2013.

[21] He J., Verstak A., Watson L.T., and Sosonkina M., "Performance Modeling and Analysis of a Massively Parallel DIRECT: Part 1", *International Journal of High Performance Computing Applications*, Vol. 23(1), pp. 14–28, Feb. 2009.

[22] He J., Verstak A., Watson L.T., and Sosonkina M., "Performance Modeling and Analysis of a Massively Parallel DIRECT: Part 2", *International Journal of High Performance Computing Applications*, Vol. 23(1), pp. 29–41, Feb. 2009.

[23] He J. et al., "Globally Optimal Transmitter Placement for Indoor Wireless Communication Systems", in *IEEE Transactions on Wireless Communications*, Vol. 3(6), pp. 1906–1911, Nov. 2004.

[24] He. J, Watson L. T., and Sosonkina M., "Algorithm 897: VTDIRECT95: Serial and Parallel Codes for the Global Optimization Algorithm DIRECT", *ACM Transactions on Mathematical Software (TOMS)*, Vol. 36(3), Article No. 17, July 2009.

[25] Jones D. R., Perttunen C. D., and Stuckman B. E., "Lipschitzian Optimization without the Lipschitz Constant", *Journal of Optimization Theory and Application*, Vol. 79(1), pp. 157–181, October 1993.

[26] Jrad M., *Multidisciplinary Optimization and Damage Tolerance of Stiffened Structures*, Ph.D. thesis, Department of Aerospace and Ocean Engineering, Virginia Polytechnic Institute & State University, Blacksburg, VA, 2015.

[27] Jrad M., Khan A. I., and Kapania R. K., "Buckling analysis of curvilinearly stiffened composite panels with cracks", in *AIAA 2014-0165, 55th Structures, Structural Dynamics, and Materials Conference (SDM)*, Maryland, National Harbor, Jan. 2014.

[28] Kolonay R. M., "Physics-Based Distributed Collaborative Design for Aerospace Vehicle Development and Technology Assessment", in *20th ISPE International Conference on Concurrent Engineering*, Melbourne, Australia, Sept. 2013.

[29] Kolonay R. M., Roberts R. W., and Lambe L.A., "A Comparison of Four Approximation Techniques for an Euler Based Induced Drag Function", in *AIAA 2008-5801, 12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, Victoria, British Columbia, Canada, Sept. 2008.

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

[31] Kolonay R. M., Thompson E. D., Camberos J. A., and Eastep F., "Active Control of Transpiration Boundary Conditions for Drag Minimization with an Euler CFD Solver", in *AIAA 2007-1891, 48th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Honolulu, Hawaii, April 2007.

[32] Kodiyalam S., Yang R. J., Gu L., and Tho C. H., "Multidisciplinary Design Optimization of a Vehicle System in a Scalable, High Performance Computing Environment", *Structural and Multidisciplinary Optimization*, Vol. 26(3/4), pp. 256–263, Feb. 2004.

[33] Liang S., *The Java Native Interface: Programmer's Guide and Specification*, Addison Wesley Longman Inc, MA, June 1999.

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

[35] Liu Q., Jrad M., Mulani S. B., and Kapania R. K., "Integrated Global Wing and Local Panel Optimization of Aircraft Wing", in *AIAA 2015-0137, 56th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Kissimmee, Florida, Jan. 2015.

[36] Ljungberg K., Holmgren S., and Carlborg O., "Simultaneous Search for Multiple QTL using the Global Optimization Algorithm DIRECT", *Bioinformatics, Vol. 20(12), pp. 1887–1895*, March 2004.

[37] Locatelli D., Liu Q., Tamijani A. Y., Mulani S. B., and Kapania R. K., "Multidisciplinary optimization of supersonic wing structures using curvilinear spars and ribs (SpaRibs)", in *AIAA 2013-1931, 54th AIAA/ASME/ASCE/AHS/ASC Structures Conference*, Boston, Massachusetts, April 2013.

[38] Mehmood A., Akhtar I., Ghommem M., Hajj M. R., and Watson L. T., "Optimization of Drag Reduction on a Cylinder Undergoing Rotary Oscillations", in *AIAA 2011-1997, 52nd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, Denver, Colorado, April 2011.

[39] Mulani S. B., Locatelli D., and Kapania R. K., "Algorithm development for optimization of arbitrary geometry panels using curvilinear stiffeners", in *AIAA 2010-2674, 51st AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, Orlando, Florida, April 2010.

[40] Mulani S. B., Slemp W. C. H., and Kapania R. K., "EBF3PanelOpt: An optimization framework for curvilinear blade-stiffened panels", *Thin Walled Structures*, Vol. 63, pp. 13–26, Feb. 2013.

[41] Papazoglou M. P., Traverso P., Dustdar S., and Leymann F., "Service-Oriented Computing: State of the Art and Research Challenges", *Computer*, Vol. 40(11), pp. 38–45, Nov. 2007.

[42] Panning T. D., Watson L. T., Allen N. A., Chen K. C., Shaffer C. A., and Tyson J. J., "Deterministic Parallel Global Parameter Estimation for a Model of the Budding Yeast Cell Cycle", *Journal of Global Optimization, Vol. 409(4), pp. 719–738*, April 2008.

[43] Raymer D. P., *Aircraft Design: A Conceptual Approach*, AIAA Education Series, New York, NY, 2006.

[44] Sobolewski M., "SORCER: Computing and Metacomputing Intergrid", in *10th International Conference on Enterprise Information Systems*, Barcelona, Spain, June 2008.

[45] Sobolewski M., *Object-Oriented Service Clouds for Transdisciplinary Computing, Cloud Computing and Services Sciences*, Springer, March 2012.

[46] Sobolewski M., "Federated Collaborations with Exertions", in *17h IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE)*, Rome, Italy, June 2008.

[47] Sobolewski M., Burton S., and Kolonay R. M., "Parametric Mogramming with Var-Oriented Modeling and Exertion-Oriented Programming Languages", in *20th ISPE International Conference on Concurrent Engineering*, Melbourne, Australia, Sept. 2013.

[48] Sobolewski M. and Kolonay R. M., "Service-Oriented Programming for Design Space Exploration", in *19th ISPE International Conference on Concurrent Engineering*, Trier, Germany, Sept. 2012.

[49] Sobolewski M. and Kolonay R. M., "Unified Mogramming with Var-Oriented Modeling and Exertion-Oriented Programming Languages", *International Journal of Communications, Network and System Sciences*, Vol. 5(9A), pp. 579–592, Sept. 2012.

[50] Sobolewski M. and Kolonay R. M., "The Convergence of Three Languages for Transdisciplinary Computing", available online at http://sorcersoft.org/publications/papers/2011/ce2011.pdf, 2011.

[51] Taminger K. M. B., and Hafley R. A., "Electron Beam Freeform Fabrication: A Rapid Metal Deposition Process", in *Proceedings of the Third Annual Automotive Composites Conference*, Troy, Michigan, Sept. 2003.

[52] Thain D., Tannenbaum T., and Livny M., "Distributed Computing in Practice: The Condor Experience", *Journal of Concurrency and Computation: Practice & Experience*, Vol. 17(2–4), pp. 323–356, Feb. 2005.

[53] Xu W., Cha J., and Sobolewski M., "A Service-Oriented Collaborative Design Platform for Concurrent Engineering", *Advanced Materials Research*, Vol. 44–46, pp. 717–724, 2008.

[54] Zhu H. and Bogy D. B., "DIRECT Algorithm and its Application to Slider Air-Bearing Surface Optimization", in *IEEE Transactions on Magnetics*, Vol. 38(5), pp. 2168–2170, Sept. 2002.

[55] Zwolak J. W., Tyson J. J., and Watson L. T., "Parameter Estimation for a Mathematical Model of the Cell Cycle in Frog Eggs", *Journal of Computational Biology*, Vol. 12(1), pp. 48–63, Feb. 2005.

Appendix A: JNI Wrappers for VTDIRECT95 and QNSTOP

As discussed in Chapter 3.1., a JVM implementation is embedded in the native code to gain access to Java classes and features. The following snippet of code illustrates the interface in native code. Implementation of the procedures defined in the interface allows invocation of a JVM and access to the Java method that evaluates the objective function.

```
! INITIALIZE is the name of the function procedure that allows
!   the initialization of a JVM.
!
! objS is the name of the real function procedure defining the
!   objective function f(x) to be minimized. objS(c, c_size)
!   returns the value f(c). c_size represents the size of 'c'.
!
INTERFACE
  FUNCTION INITIALIZE() BIND(C, NAME='initialize')
    USE, INTRINSIC :: ISO_C_BINDING, ONLY: C_PTR, C_INT
    IMPLICIT NONE
    INTEGER(C_INT):: INITIALIZE
  END FUNCTION INITIALIZE

  FUNCTION  objS(c, c_size) RESULT(f) BIND(C, NAME='objS')
    USE, INTRINSIC :: ISO_C_BINDING, ONLY: C_DOUBLE, C_PTR, C_INT
    TYPE(C_PTR), INTENT(IN), VALUE :: c
    INTEGER(C_INT), INTENT(IN):: c_size
    REAL(KIND = C_DOUBLE) :: f
  END FUNCTION objS
END INTERFACE
```

The following function `objfuncS` calls the corresponding glue code in C which in turn allows access to the objective function in Java via the invocation interface.

```
FUNCTION objfuncS(c, iflag) RESULT(f)
! On input:
! c     - Point coordinates.
!
! On output:
! iflag  - A flag that is used to indicate the status of the
!        function evaluation. It is 0 for normal status.
! f     - Function value at 'c'.
!
USE REAL_PRECISION, ONLY : R8
USE, INTRINSIC :: ISO_C_BINDING
IMPLICIT NONE

! Dummy variables.
REAL(KIND = R8), DIMENSION(:), INTENT(IN):: c
INTEGER, INTENT(OUT):: iflag
REAL(KIND = R8) :: f

! Local variables.
REAL(KIND = C_DOUBLE), TARGET :: designPoints(1:SIZE(c))
TYPE(C_PTR) :: designPointsPtr
INTEGER(C_INT) :: N
```

```
! objS is the name of the real function procedure in C defining the
!   objective function f(x) to be minimized. objS(c, c_size)
!   returns the value f(c). c_size represents the size of ‘c’.
!
INTERFACE
  FUNCTION  objS(c, c_size) RESULT(f) BIND(C, NAME=‘objS’)
    USE, INTRINSIC :: ISO_C_BINDING, ONLY: C_DOUBLE, C_PTR, C_INT
    TYPE(C_PTR), INTENT(IN), VALUE :: c
    INTEGER(C_INT), INTENT(IN) :: c_size
    REAL(KIND = C_DOUBLE) :: f
  END FUNCTION objS
END INTERFACE

! Conversion of assumed-shape array to static array.
designPointsPtr = c_loc(designPoints(1))
designPoints = c
N = SIZE(c)

! Call to glue-code in C.
f = objS(designPointsPtr, N)

iflag = 0
RETURN
END FUNCTION objfuncS
```

The following program `objfuncGlue.c` is the glue-code in C that holds Fortran and Java together.

```
#include <stdio.h>
#include <stdlib.h>
#include <jni.h>

extern double objS(double *, int *);
extern int initialize();
/* denotes a Java VM */
static JavaVM *jvm;

JNIEnv* create_vm(JavaVM **jvm)
{
   /* Pointer to native method interface. */
   JNIEnv* env;
   /* JDK/JRE VM initialization arguments. */
   JavaVMInitArgs args;
   JavaVMOption options;
   args.version = JNI_VERSION_1_8;
   args.nOptions = 1;
   options.optionString = "-Djava.class.path=./";
   args.options = &options;
   args.ignoreUnrecognized = 0;
   int flag;
   /*
     Load and initialize a Java VM, return a JNI interface pointer in env.
    */
   flag = JNI_CreateJavaVM(jvm, (void**)&env, &args);
```

```
    if (flag < 0 || !env)
       printf("Unable to Launch JVM %d",flag);
    else
       printf("Launched JVM successfully");
    return env;
}
int initialize()
{
JNIEnv *env_init;
/* Call to create a JVM. */
env_init = create_vm(&jvm);
if(env_init == NULL)
  return 1;
}
double objS(double *c_ptr, int *c_size) {
    /* JNI interface pointer. */
    JNIEnv *env;
    /* Local variables. */
    jdouble *dvarsPtr;
    jdoubleArray dvars;
    jint size;
    double f;

    /* Dereference pointers. */
    size = *c_size;
    dvarsPtr = (double *)c_ptr;

    /*
       Allow the current thread to attach itself to the JVM and obtain a
       JNI interface pointer.
     */
    jint flag = (*jvm)->AttachCurrentThread(jvm, (void **)&env, NULL);
     if (flag < 0 || !env)
        printf("Unable to attach to the existing instance of JVM %d",flag);
        exit(1);
    else
        printf("Attached to the existing JVM thread");

    /* Construct a new primitive array object. */
    dvars = (*env)->NewDoubleArray(env, size);
    (*env)->SetDoubleArrayRegion(env, dvars, 0, size, (const
    jdouble*)dvarsPtr);

    /* Local variables for invocation of Java classes and methods. */
    jclass objFuncJ_class;
    jmethodID main_method;
    jmethodID evaluate_method;

    /* Find Java class by name. */
    objFuncJ_class = (*env)->FindClass(env, "objFuncJ");

     /* Find Java method by name and signature. */
    evaluate_method = (*env)->GetStaticMethodID(env, objFuncJ_class,
    "evaluate", "([D)D");

    /* Invoke the objFuncJ.evaluate method using the JNI. */
```

```
   f = (*env)->CallStaticDoubleMethod(env, objFuncJ_class, evaluate_method,
   dvars);

   /* Return function value. */
   return f;
}
```

The following program `objfuncJ.java` contains the objective function for the subroutines VTdirect, pVTdirect, QNSTOPS, and QNSTOPP.

```
import java.io.*;
import java.lang.*;
import java.util.*;

public class objfuncJ
{
   public static double evaluate(double[] designPoints) {
      /* Local variables */
      double mass =0;
      File desVars = null;
      File respVars = null;

      try {
         String desVarsFilename = "dvar" + UUID.randomUUID() + ".vef";
         desVars = new File(desVarsFilename);
         String respVarsFilename = "resp" + UUID.randomUUID() + ".vef";
         respVars = new File(respVarsFilename);

         /* Create files for input to the EBF3PanelOpt framework. */
         if (!desVars.exists()) {
            desVars.createNewFile();
         }
         if (!respVars.exists()) {
            respVars.createNewFile();
         }
         FileWriter desVarsWriter = new
         FileWriter(desVars.getAbsoluteFile());
         BufferedWriter bw = new BufferedWriter(desVarsWriter);

         /* Write design points to a file. */
         for(int k=0; k<designPoints.length; k++) {
            bw.write(""+String.format("%.12f",designPoints[k]));
         }
         bw.close();
      } catch (IOException e) {
         e.printStackTrace();
         System.exit(-1);
      }

      try {
         /* System call to the EBF3PanelOpt framework in Python. */
         Process panelOpt = Runtime.getRuntime().exec("python
         evaluate_response.py -i stiffened_plate_input_data.txt -idv "
         + desVars + " -o " + respVars);
         panelOpt.waitFor();
```

```
        } catch (Exception e) {
            e.printStackTrace();
            System.exit(-1);
        }

        try {
            if (!respVars.exists()) {
                System.out.println("ERROR");
                System.exit(-1);
        }
            /* Read response from file. */
            FileReader respVarsReader = new
            FileReader(respVars.getAbsoluteFile());
            BufferedReader br = new BufferedReader(respVarsReader);
            mass = br.readLine();
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(-1);
        }
        /* Return response to glue code in C. */
        return mass;
    }
}
```

Modified JNI Wrapper for QNSTOPP

As discussed in Chapter 3.5.2, 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. The following code snippet illustrates the modified loop over the experimental design samples.

```
counter = 0
EvalLoop: DO I=1,LN
   bin(P*counter+1:I*P)=(LB + BoxDiag*XValues(1:P,I))
   counter = counter + 1
  END DO EvalLoop
```

The one-dimensional array `bin` is the concatenation of the design points chunked together to be used in function evaluation calls to SORCER. The procedure that allows access to the Java method that evaluates the objective function is modified as follows. In this case, the Java method returns an array of function evaluation values. A dummy array is passed from the Fortran 95 subroutine to Java, which is populated with the function evaluation values in the Java method. It is specified that the dummy argument will be used to pass data from the Java method to the calling native program. The dummy argument is undefined on entry and is defined before it is referenced in the subroutine.

A JVM implementation is embedded in the native code to gain access to Java classes and features. The following snippet of code illustrates the interface in native code. Implementation of the procedures defined in the interface allows invocation of a JVM and access to the Java method that evaluates the objective function.

```
! INITIALIZE is the name of the function procedure that allows
!   the initialization of a JVM.
!
! objP is the name of the procedure in C defining the objective
!   function f(x) for each 'x' in 'c' to be minimized.
!   objP(c, c_size, p, resp) sets the function values in the
!   array 'resp'. 'c' contains multiple design points chunked
!   together. 'c_size' represents the size of 'c'. 'p' is the
!   problem dimension.
!
INTERFACE
  FUNCTION INITIALIZE() BIND(C, NAME='initialize')
    USE, INTRINSIC :: ISO_C_BINDING, ONLY: C_PTR, C_INT
    IMPLICIT NONE
    INTEGER(C_INT):: INITIALIZE
  END FUNCTION INITIALIZE

  SUBROUTINE objP(c, c_size, p, resp) bind (C, name = "objP")
    USE, INTRINSIC :: ISO_C_BINDING
    IMPLICIT NONE
    TYPE(C_PTR), INTENT(IN), VALUE :: c
    INTEGER(C_INT), INTENT(IN):: c_size
    INTEGER(C_INT), INTENT(IN) :: p
    TYPE(C_PTR), INTENT(IN), VALUE :: resp
  END SUBROUTINE objP
```

```
END INTERFACE
```

The following function `objfuncP` calls the corresponding glue code in C which in turn allows access to the objective function in Java via the invocation interface.

```
FUNCTION objfuncP(c, p, resp) RESULT(iflag)
! On input:
! c      - Chunked point coordinates.
! p      - Problem dimension.
! resp    - Dummy array.
!
! On output:
! resp   - Array containing the function values.
! iflag  - A flag that is used to indicate the status of the
!        function evaluation values returned from Java. It is
!        0 for normal status.
!
USE REAL_PRECISION, ONLY : R8
USE, INTRINSIC :: ISO_C_BINDING
IMPLICIT NONE

! Dummy variables.
REAL(KIND = R8), DIMENSION(:), INTENT(IN) :: c
INTEGER, INTENT(IN) :: p
REAL(KIND = R8), DIMENSION(:), INTENT(INOUT) :: resp
INTEGER :: iflag

! Local variables.
REAL(KIND = C_DOUBLE), TARGET :: designPoints(1:SIZE(c))
TYPE(C_PTR) :: designPointsPtr
REAL(KIND = C_DOUBLE), TARGET :: respVars(1:SIZE(resp))
TYPE(C_PTR) :: respVarsPtr

! objP is the name of the procedure in C defining the objective
!   function f(x) for each 'x' in 'c' to be minimized.
!   objP(c, c_size, p, resp) sets the function values in the
!   array 'resp'. 'c' contains multiple design points chunked
!   together. 'c_size' represents the size of 'c'. 'p' is the
!   problem dimension.
!
INTERFACE
  SUBROUTINE objP(c, c_size, p, resp) bind (C, name = "objP")
    USE, INTRINSIC :: ISO_C_BINDING
    IMPLICIT NONE
    TYPE(C_PTR), INTENT(IN), VALUE :: c
    INTEGER(C_INT), INTENT(IN):: c_size
    INTEGER(C_INT), INTENT(IN) :: p
    TYPE(C_PTR), INTENT(IN), VALUE :: resp
  END SUBROUTINE objP
END INTERFACE

! Conversion of assumed-shape arrays to static arrays.
designPointsPtr = c_loc(designPoints(1))
designPoints = c
respVarsPtr = c_loc(respVars(1))
```

```
! Call to glue-code in C
CALL objP(designPointsPtr, SIZE(c), p, respVarsPtr)

! Copy contents of array returned by C to array 'resp'.
resp = respVars

iflag = 0
RETURN
END FUNCTION objfuncP
```

The following procedure is added to glue code in C.

```
void objP(double *c_ptr, int *c_size, int *p, double *r_ptr) {
   /* JNI interface pointer. */
   JNIEnv *env;
   /* Local variables. */
   jdouble *dvarsPtr;
   jdouble *respPtr;
   jdoubleArray dvars;
   jint size;
   jint P;
   int k;
   int length;
   double *respVars;

   /* Dereference pointers. */
   size = *c_size;
   dvarsPtr = (double *)c_ptr;
   respPtr = (double *)resp_ptr;
   P = *p;

   /*
      Allow the current thread to attach itself to the JVM and obtain a
      JNI interface pointer.
    */
   jint flag = (*jvm)->AttachCurrentThread(jvm, (void **)&env, NULL);
    if (flag < 0 || !env)
       printf("Unable to attach to the existing instance of JVM %d",flag);
       exit(1);
   else
       printf("Attached to the existing JVM thread");

   /* Construct a new primitive array object. */
   dvars = (*env)->NewDoubleArray(env, size);
   (*env)->SetDoubleArrayRegion(env, dvars, 0, size, (const
   jdouble*)dvarsPtr);

   /* Local variables for invocation of Java classes and methods. */
   jclass objFuncJ_class;
   jmethodID main_method;
   jmethodID evaluate_method;

   /* Find Java class by name. */
   objFuncJ_class = (*env)->FindClass(env, "objFuncJ");
```

```
    /* Find Java method by name and signature. */
   evaluate_method = (*env)->GetStaticMethodID(env, objFuncJ_class,
   "evaluateP", "([DI)[D");

   /* Invoke the objFuncJ.evaluate method using the JNI. */
   jdoubleArray respArray = (*env)->CallStaticObjectMethod(env,
   objFuncJ_class, evaluate_method, dvars, P);
   if(respArray != NULL) {
      length = (*env)->GetArrayLength(env, respArray);
      respVars = (*env)->GetDoubleArrayElements(env, respArray, NULL);
   }
   else
   for (k = 0; k < length; k++) {
      respPtr[k] = respVars[k];
   }

   /* Release elements of the array. */
   (*env)->ReleaseDoubleArrayElements(env, respArray, value, 0);
   return ;
}
```

Appendix B: VTdirect as a SORCER service

The following method `VtdirectProviderImpl.java` is an analysis provider that provides VTdirect as a service over the SORCER network, and is implemented in accordance with principles of exertion-oriented programming (EOP) as described in Chapter 3.2.

```
/*
  Distribution Statement:
  This computer software has been developed under sponsorship of the United
  States Air Force Research Lab. Any further distribution or use by anyone
  or any data contained therein, unless otherwise specifically provided for,
  is prohibited without the written approval of AFRL/RQVC-MSTC, 2210 8th
  Street Bldg 146, Room 218, WPAFB, OH  45433.

  Disclaimer:
  This material was prepared as an account of work sponsored by an agency of
  the United States Government. Neither the United States Government nor the
  United States Air Force, nor any of their employees, makes any warranty,
  express or implied, or assumes any legal liability or responsibility for
  the accuracy, completeness, or usefulness of any information, apparatus,
  product, or process disclosed, or represents that its use would not infringe
  privately owned rights.
*/

package mil.afrl.mstc.open.vtdirect.provider;

import com.sun.jini.start.LifeCycle;
import mil.afrl.mstc.open.vtdirect.Vtdirect;
import mil.afrl.mstc.open.vtdirect.VtdirectContext;
import mil.afrl.mstc.open.core.provider.EngineeringProvider;
import sorcer.service.Context;
import sorcer.util.GenericUtil;
import java.io.File;
import java.net.URL;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;
import java.util.logging.Level;
import java.util.logging.Logger;
/*
  Implementation of the {mil.afrl.mstc.open.vtdirect.Vtdirect} interface.

  @author MSTC Engineering Project Generator
*/
public class VtdirectProviderImpl extends EngineeringProvider implements
Vtdirect {
   private final static Logger logger =
   Logger.getLogger(VtdirectProviderImpl.class.getName());

   /*
     Default constructor, initializes the provider with the properties file
     loaded as a resource.
     @throws Exception if initialization fails.
```

```
*/
public VtdirectProviderImpl() throws Exception {
   this("vtdirect/provider.properties");
}

/*
  Create the provider using a properties file.
  @param propFile: The properties file, must not be {@code null}.
  @throws Exception: If initialization fails.
*/
public VtdirectProviderImpl(final String propFile) throws Exception {
   super(propFile);
   initialize();
}

/*
  Create the provider if called from the River
  {com.sun.jini.start.ServiceStarter} approach.
  @param args: Parameters that point to configuration file and possible
  overrides. Never {@code null}.
  @param lifeCycle: The lifecycle container to be notified if the
  provider terminates.
  @throws Exception if initialization fails.
*/
public VtdirectProviderImpl(final String[] args, final LifeCycle lifeCycle)
throws Exception {
   super(args, lifeCycle);
   initialize();
}

 /* Provides provider specific initialization and environment checking. */
private void initialize() {

}

public Context execute(final Context context) {
   /* Log status in a log file. */
   logger.info("Running execute");
   File scratchDir = null;
   URL scratchUrl = null;
   String serviceIdString = doThreadMonitor(null);

   try {
      /* Cast to domain-specific context. */
      VtdirectContext vtdirectContext = (VtdirectContext) context;

      /* Get scratch directory. */
      scratchDir = getScratchDir(context, "vtdirect_execute_");
      scratchUrl = getScratchURL(scratchDir);

      /* Download input file for VTdirect and model input file. */
      URL vtdirectInputUrl = (URL) vtdirectContext.getVTdirectInput();
      File vtdirectInputFile = new File(scratchDir, "direct.nml");
      URL vtdirectModelUrl = (URL) vtdirectContext.getModelInput();
      File vtdirectModelFile = new File(scratchDir, "modelInfo.txt");
      GenericUtil.download(vtdirectInputUrl, vtdirectInputFile);
```

```
            GenericUtil.download(vtdirectModelUrl, vtdirectModelFile);

            /* Get system properties. */
            String vtdirectExec = null;
            if (GenericUtil.isLinux()) vtdirectExec =
            getProperty("provider.exec.vtdirect.linux64").trim();
            if (GenericUtil.isMac()) vtdirectExec =
            getProperty("provider.exec.vtdirect.mac64").trim();

            /* Create script to run VTdirect. */
            File nativeHome = new File(vtdirectExec).getParentFile();
            Vector<String> script = new Vector<String>();
            script.add("#!/bin/bash");
            script.add("cd " + scratchDir.getAbsolutePath());
            script.add("cp " + new File(vtdirectExec).getAbsolutePath() + " .");
            script.add("cp " + nativeHome + "/vtdirectObjFunc.class .");
            script.add("cp " + nativeHome + "/vtdirect .");
            script.add("./" + new File(vtdirectExec).getName() + " &>
            vtdirectStdOutAndError.txt");

            /* Run shell script. */
            File scriptFile = new File(scratchDir, "script.sh");
            GenericUtil.setFileContents(scriptFile, script);
            scriptFile.setExecutable(true);
            Process p = Runtime.getRuntime().exec(new String[]"sh", "-c",
            scriptFile.getAbsolutePath());
            p.waitFor();

            /* Add output to Context. */
            File vtdirectoutfile1 = new File(scratchDir, "directOutput.txt");
            URL vtdirectOutputFile = this.getScratchURL(vtdirectoutfile1);
            vtdirectContext.setOutput(vtdirectOutputFile);
            logger.info("vtdirectContext = " + vtdirectContext);
        } catch (Exception e) {
            StringBuilder reason = new StringBuilder();
            reason.append("vtdirectProviderImpl caught exception =
            ").append(e.getClass().getName())
                   .append("scratchDir = ").append(scratchDir)
                   .append("scratchUrl = ").append(scratchUrl);
            context.reportException(reason.toString(), e,
            getProviderInfo("execute"));
            logger.log(Level.SEVERE, reason.toString(), e);
            return context;
        }
        logger.info("Returning from service execute; Output context =
        "+context);
        return context;
    }
}
```

The following method `VtdirectRequestor.java` is a requestor that creates exertions and submits them to the grid in order to utilize services provided by the corresponding analysis provider.

```
/*
  Distribution Statement:
```

```
  This computer software has been developed under sponsorship of the United
  States Air Force Research Lab. Any further distribution or use by anyone
  or any data contained therein, unless otherwise specifically provided for,
  is prohibited without the written approval of AFRL/RQVC-MSTC, 2210 8th
  Street Bldg 146, Room 218, WPAFB, OH  45433.

  Disclaimer:
  This material was prepared as an account of work sponsored by an agency of
  the United States Government. Neither the United States Government nor the
  United States Air Force, nor any of their employees, makes any warranty,
  express or implied, or assumes any legal liability or responsibility for
  the accuracy, completeness, or usefulness of any information, apparatus,
  product, or process disclosed, or represents that its use would not infringe
  privately owned rights.
*/
package mil.afrl.mstc.open.vtdirect.requestor;

import mil.afrl.mstc.open.data.DataService;
import mil.afrl.mstc.open.requestor.EngineeringRequestor;
import mil.afrl.mstc.open.vtdirect.Vtdirect;
import org.sorcer.core.requestor.SorcerTester;
import sorcer.core.exertion.NetTask;
import sorcer.core.requestor.SorcerRequestor;
import sorcer.core.signature.NetSignature;
import sorcer.service.Exertion;
import sorcer.util.GenericUtil;
import sorcer.util.Sorcer;
import java.io.*;
import java.lang.*;
import java.net.URL;
import java.util.Properties;
import java.util.Vector;
import java.util.logging.Logger;
import mil.afrl.mstc.open.vtdirect.VtdirectContext;
/*
  A requestor that uses the following provider:
  {mil.afrl.mstc.open.vtdirect.Vtdirect}.

  @author MSTC Engineering Project Generator
*/
public class VtdirectRequestor extends SorcerRequestor {
   private static Logger logger =
   Logger.getLogger(VtdirectRequestor.class.getName());
   private File dataDir = new File(System.getProperty("data.dir"));
   private DataService dataService;
   private Properties properties;
   private EngineeringRequestor engineeringRequestor = new
   EngineeringRequestor();
   /*
     Run the requestor.
     @param args: Arguments to process.
     @throws Exception if there are problems running the requestor.
   */
   public void run(String... args) throws Exception {
      File engHome = new File(System.getProperty("eng.home"));
      File dataDir = new File(engHome, "data");
```

```
File tempDir = new File(dataDir, "temp");

/* Get input file for VTdirect and model input file. */
File vtdirectInputFile = new
File(engHome,"/optimization/vtdirect/vtdirect-req/data/direct.nml");
File modelInputFile = new
File(engHome,"/optimization/vtdirect/vtdirect-req/data/modelInfo.txt");

/* Edit modeInputFile to adjust model name for user. */
Vector<String> modelInputFileContents =
GenericUtil.getFileContents(modelInputFile);
GenericUtil.printVect(modelInputFileContents);
modelInputFileContents.setElementAt(Sorcer.getActualName
(modelInputFileContents.elementAt(0).trim()), 0);
GenericUtil.printVect(modelInputFileContents);
File modelInputFileTemp = new File(tempDir, "modelInfo.txt");
GenericUtil.setFileContents(modelInputFileTemp,
modelInputFileContents);

/* Copy input files to the scratch directory. */
URL vtdirectInputUrl =
SorcerTester.copyFileToScratchAndGetUrl(vtdirectInputFile, dataDir);
URL modelInputUrl =
SorcerTester.copyFileToScratchAndGetUrl(modelInputFileTemp, dataDir);

/* Add input to Context. */
VtdirectContext context = new VtdirectContext("VTdirectContext");
context.setVTdirectInput(vtdirectInputUrl);
context.setModelInput(modelInputUrl);

/* Construct execute method. */
String providerName = Sorcer.getActualName("Engineering-Vtdirect");
String serviceName = "execute";
NetSignature methodEN = new NetSignature(serviceName,
          Vtdirect.class, providerName);

/* Construct Task. */
NetTask vtdirectTask = new NetTask("run execute", "Task to run
VTDIRECT95", methodEN);

/* Put the Context into the task. */
vtdirectTask.setContext(context);
logger.info("scott1Task = " + vtdirectTask);

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

logger.info("Returned Task after exert: " + result);
logger.info("Context output =
"+((VtdirectContext)result.getContext()).getOutput());

/* Read output from Context. */
URL vtdirectOutputUrl = (URL) ((VtdirectContext)
result.getContext()).getOutput();
Vector<String> vtdirectOutputFileContents =
GenericUtil.getFileContents(vtdirectOutputUrl);
```

```
        GenericUtil.printVect(vtdirectOutputFileContents);
    }
}
```

Appendix C: QNSTOPS as a SORCER service

The following method `QnstopsProviderImpl.java` is an analysis provider that provides QNSTOPS as a service over the SORCER network, and is implemented in accordance with principles of exertion-oriented programming (EOP) as described in Chapter 3.2.

```
/*
  Distribution Statement:
  This computer software has been developed under sponsorship of the United
  States Air Force Research Lab. Any further distribution or use by anyone
  or any data contained therein, unless otherwise specifically provided for,
  is prohibited without the written approval of AFRL/RQVC-MSTC, 2210 8th
  Street Bldg 146, Room 218, WPAFB, OH  45433.

  Disclaimer:
  This material was prepared as an account of work sponsored by an agency of
  the United States Government. Neither the United States Government nor the
  United States Air Force, nor any of their employees, makes any warranty,
  express or implied, or assumes any legal liability or responsibility for
  the accuracy, completeness, or usefulness of any information, apparatus,
  product, or process disclosed, or represents that its use would not infringe
  privately owned rights.
*/
package mil.afrl.mstc.open.qnstops.provider;

import com.sun.jini.start.LifeCycle;
import mil.afrl.mstc.open.qnstops.Qnstops;
import mil.afrl.mstc.open.qnstops.QnstopsContext;
import mil.afrl.mstc.open.core.provider.EngineeringProvider;
import sorcer.service.Context;
import sorcer.util.GenericUtil;
import java.io.File;
import java.net.URL;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.Vector;

/*
  Implementation of the {mil.afrl.mstc.open.qnstops.Qnstops} interface.
  @author MSTC Engineering Project Generator
*/
public class QnstopsProviderImpl extends EngineeringProvider implements
Qnstops {
    private final static Logger logger =
    Logger.getLogger(QnstopsProviderImpl.class.getName());

    /*
      Default constructor initializes the provider with the properties file
      loaded as a resource.
      @throws Exception if initialization fails.
    */
```

```
    public QnstopsProviderImpl() throws Exception {
       this("qnstops/provider.properties");
    }

    /*
      Create the provider using a properties file.
      @param propFile: The properties file, must not be {@code null}.
      @throws Exception if initialization fails.
    */
    public QnstopsProviderImpl(final String propFile) throws Exception {
       super(propFile);
       initialize();
    }

    /*
      Create the provider if called from the River
      {com.sun.jini.start.ServiceStarter} approach.
      @param args: Parameters that point to configuration file and possible
      overrides. Never {@code null}.
      @param lifeCycle: The lifecycle container to be notified if the
      provider terminates.
      @throws Exception if initialization fails.
    */
    public QnstopsProviderImpl(final String[] args, final LifeCycle lifeCycle)
    throws Exception {
       super(args, lifeCycle);
       initialize();
    }

    /* Provides provider specific initialization and environment checking. */
    private void initialize() {

    }

    public Context execute(final Context context) {
/* Logs status in a log file */
       logger.info("Running execute");
       File scratchDir = null;
       URL scratchUrl = null;
       String serviceIdString = doThreadMonitor(null);
       try {
          /* Cast to domain-specific context. */
          QnstopsContext qnstopsContext = (QnstopsContext) context;

          /* Get scratch directory. */
          scratchDir = getScratchDir(context, "qnstops_execute_");
          scratchUrl = getScratchURL(scratchDir);

          /* Download input file for QNSTOPS and model input file. */
          URL qnstopsInputUrl = (URL) qnstopsContext.getQNSTOPSInput();
          File qnstopsInputFile = new File(scratchDir, "qnstops.nml");
          URL qnstopsModelUrl = (URL) qnstopsContext.getModelInput();
          File qnstopsModelFile = new File(scratchDir, "modelInfo.txt");
          GenericUtil.download(qnstopsInputUrl, qnstopsInputFile);
          GenericUtil.download(qnstopsModelUrl, qnstopsModelFile);
```

```
            /* Get system properties. */
            String qnstopsExec = null;
            if (GenericUtil.isLinux()) qnstopsExec =
            getProperty("provider.exec.qnstops.linux64").trim();
            if (GenericUtil.isMac()) qnstopsExec =
            getProperty("provider.exec.qnstops.mac64").trim();

            /* Create script to run QNSTOPS. */
            File nativeHome = new File(qnstopsExec).getParentFile();
            Vector<String> script = new Vector<String>();
            script.add("#!/bin/bash");
            script.add("cd " + scratchDir.getAbsolutePath());
            script.add("cp " + new File(qnstopsExec).getAbsolutePath() + " .");
            script.add("cp " + nativeHome + "/qnstopsObjFunc.class .");
            script.add("cp " + nativeHome + "/qnstops .");
            script.add("./" + new File(qnstopsExec).getName() + " &>
            qnstopsStdOutAndError.txt");

            /* Run shell script. */
            File scriptFile = new File(scratchDir, "script.sh");
            GenericUtil.setFileContents(scriptFile, script);
            scriptFile.setExecutable(true);
            Process p = Runtime.getRuntime().exec(new String[]"sh", "-c",
            scriptFile.getAbsolutePath());
            p.waitFor();

            /* Add output to Context. */
            File qnstopsoutfile1 = new File(scratchDir, "QNSTOPSOutput.txt");
            URL qnstopsOutputFile = this.getScratchURL(qnstopsoutfile1);
            qnstopsContext.setOutput(qnstopsOutputFile);
            logger.info("QNSTOPSContext = " + qnstopsContext);
        } catch (Exception e) {
            StringBuilder reason = new StringBuilder();
            reason.append("error: qnstopsProviderImpl caught exception =
            ").append(e.getClass().getName())
                   .append("scratchDir = ").append(scratchDir)
                   .append("scratchUrl = ").append(scratchUrl);
            context.reportException(reason.toString(), e,
            getProviderInfo("execute"));
            logger.log(Level.SEVERE, reason.toString(), e);
            return context;
        }
        logger.info("Returning from service execute; Output context =
        "+context);
        return context;
    }
}
```

The following method `QnstopsRequestor.java` is a requestor that creates exertions and submits them to the grid in order to utilize services provided by the corresponding analysis provider.

```
/*
  Distribution Statement:
  This computer software has been developed under sponsorship of the United
  States Air Force Research Lab. Any further distribution or use by anyone
```

```
  or any data contained therein, unless otherwise specifically provided for,
  is prohibited without the written approval of AFRL/RQVC-MSTC, 2210 8th
  Street Bldg 146, Room 218, WPAFB, OH  45433.

  Disclaimer:
  This material was prepared as an account of work sponsored by an agency of
  the United States Government. Neither the United States Government nor the
  United States Air Force, nor any of their employees, makes any warranty,
  express or implied, or assumes any legal liability or responsibility for
  the accuracy, completeness, or usefulness of any information, apparatus,
  product, or process disclosed, or represents that its use would not infringe
  privately owned rights.
*/
package mil.afrl.mstc.open.qnstops.requestor;

import mil.afrl.mstc.open.qnstops.Qnstops;
import mil.afrl.mstc.open.data.DataService;
import mil.afrl.mstc.open.provider.ProviderStrategy;
import mil.afrl.mstc.open.requestor.EngineeringRequestor;
import mil.afrl.mstc.open.qnstops.Qnstops;
import mil.afrl.mstc.open.qnstops.QnstopsContext;
import sorcer.core.requestor.SorcerRequestor;
import mil.afrl.mstc.open.task.TaskFactory;
import sorcer.core.context.ServiceContext;
import sorcer.core.exertion.NetTask;
import sorcer.core.signature.NetSignature;
import sorcer.service.Context;
import sorcer.service.Exertion;
import sorcer.util.Sorcer;
import sorcer.util.GenericUtil;
import org.sorcer.core.requestor.SorcerTester;
import java.io.File;
import java.util.Properties;
import java.util.logging.Logger;
import java.net.URL;
import java.util.Vector;

/*
  A requestor that uses the following provider:
  {mil.afrl.mstc.open.qnstops.Qnstops}.
  @author MSTC Engineering Project Generator
*/
public class QnstopsRequestor extends SorcerRequestor {
   private static Logger logger =
   Logger.getLogger(QnstopsRequestor.class.getName());
   private File dataDir = new File(System.getProperty("data.dir"));
   private DataService dataService; private Properties properties;
   private EngineeringRequestor engineeringRequestor = new
   EngineeringRequestor();

   /*
     Run the requestor.
     @param args: Arguments to process.  @throws Exception if there are
     problems running the requestor.
   */
   public void run(String... args) throws Exception {
```

```
File engHome = new File(System.getProperty("eng.home"));
File dataDir = new File(engHome, "data");
File tempDir = new File(dataDir, "temp");

/* Get input file for QNSTOPS and model input file. */
File qnstopsInputFile = new
File(engHome,"/optimization/qnstops/qnstops-req/data/qnstops.nml");
File modelInputFile = new
File(engHome,"/optimization/qnstops/qnstops-req/data/modelInfo.txt");

/* Edit modeInputFile to adjust model name for user. */
Vector<String> modelInputFileContents =
GenericUtil.getFileContents(modelInputFile);
GenericUtil.printVect(modelInputFileContents);
modelInputFileContents.setElementAt(Sorcer.getActualName
(modelInputFileContents.elementAt(0).trim()), 0);
GenericUtil.printVect(modelInputFileContents);
File modelInputFileTemp = new File(tempDir, "modelInfo.txt");
GenericUtil.setFileContents(modelInputFileTemp,
modelInputFileContents);

/* Copy input files to the scratch directory. */
URL qnstopsInputUrl =
SorcerTester.copyFileToScratchAndGetUrl(qnstopsInputFile, dataDir);
URL modelInputUrl =
SorcerTester.copyFileToScratchAndGetUrl(modelInputFileTemp, dataDir);

/* Add input to Context. */
QnstopsContext context = new QnstopsContext("QNSTOPSContext");
context.setQNSTOPSInput(qnstopsInputUrl);
context.setModelInput(modelInputUrl);

/* Construct execute method. */
String providerName = Sorcer.getActualName("Engineering-Qnstops");
String serviceName = "execute";
NetSignature methodEN = new NetSignature(serviceName,
       Qnstops.class,
       providerName);

/* Construct Task. */
NetTask qnstopsTask = new NetTask("run execute", "Task to run
QNSTOPS", methodEN);

/* Put the context into the task. */
qnstopsTask.setContext(context);
logger.info("qnstopsTask = " + qnstopsTask);

/* Exert collaboration. */
Exertion result = qnstopsTask.exert();

logger.info("Returned Task after exert: " + result);
logger.info("Context output =
"+((QnstopsContext)result.getContext()).getOutput());

/* Read output from Context. */
URL qnstopsOutputUrl = (URL) ((QnstopsContext)
```

```
        result.getContext()).getOutput();
        Vector<String> qnstopsOutputFileContents =
        GenericUtil.getFileContents(qnstopsOutputUrl);
        GenericUtil.printVect(qnstopsOutputFileContents);
    }
}
```

Appendix D: QNSTOPP as a SORCER service

The following method `QnstoppProviderImpl.java` is an analysis provider that provides QNSTOPP as a service over the SORCER network, and is implemented in accordance with principles of exertion-oriented programming (EOP) as described in Chapter 3.2.

```
/*
  Distribution Statement:
  This computer software has been developed under sponsorship of the United
  States Air Force Research Lab. Any further distribution or use by anyone
  or any data contained therein, unless otherwise specifically provided for,
  is prohibited without the written approval of AFRL/RQVC-MSTC, 2210 8th
  Street Bldg 146, Room 218, WPAFB, OH  45433.

  Disclaimer:
  This material was prepared as an account of work sponsored by an agency of
  the United States Government. Neither the United States Government nor the
  United States Air Force, nor any of their employees, makes any warranty,
  express or implied, or assumes any legal liability or responsibility for
  the accuracy, completeness, or usefulness of any information, apparatus,
  product, or process disclosed, or represents that its use would not infringe
  privately owned rights.
*/
package mil.afrl.mstc.open.qnstopp.provider;

import com.sun.jini.start.LifeCycle;
import mil.afrl.mstc.open.qnstopp.Qnstopp;
import mil.afrl.mstc.open.qnstopp.QnstoppContext;
import mil.afrl.mstc.open.core.provider.EngineeringProvider;
import sorcer.service.Context;
import sorcer.util.GenericUtil;
import java.io.File;
import java.net.URL;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.Vector;

/*
  Implementation of the {mil.afrl.mstc.open.qnstopp.Qnstopp} interface.
  @author MSTC Engineering Project Generator
*/
public class QnstoppProviderImpl extends EngineeringProvider implements
Qnstopp {
   private final static Logger logger =
   Logger.getLogger(QnstoppProviderImpl.class.getName());

   /*
     Default constructor initializes the provider with the properties file
     loaded as a resource.
     @throws Exception if initialization fails.
   */
```

```
    public QnstoppProviderImpl() throws Exception {
       this("qnstopp/provider.properties");
    }

    /*
      Create the provider using a properties file.
      @param propFile: The properties file, must not be {@code null}.
      @throws Exception if initialization fails.
    */
    public QnstoppProviderImpl(final String propFile) throws Exception {
       super(propFile);
       initialize();
    }

    /*
      Create the provider if called from the River
      {com.sun.jini.start.ServiceStarter} approach.
      @param args: Parameters that point to configuration file and possible
      overrides. Never {@code null}.
      @param lifeCycle: The lifecycle container to be notified if the
      provider terminates.
      @throws Exception if initialization fails.
    */
    public QnstoppProviderImpl(final String[] args, final LifeCycle lifeCycle)
    throws Exception {
       super(args, lifeCycle);
       initialize();
    }

    /* Provides provider specific initialization and environment checking. */
    private void initialize() {

    }

    public Context execute(final Context context) {
/* Logs status in a log file */
       logger.info("Running execute");
       File scratchDir = null;
       URL scratchUrl = null;
       String serviceIdString = doThreadMonitor(null);
       try {
          /* Cast to domain-specific context. */
          QnstoppContext qnstoppContext = (QnstoppContext) context;

          /* Get scratch directory. */
          scratchDir = getScratchDir(context, "qnstopp_execute_");
          scratchUrl = getScratchURL(scratchDir);

          /* Download input file for QNSTOPP and model input file. */
          URL qnstoppInputUrl = (URL) qnstoppContext.getQNSTOPPInput();
          File qnstoppInputFile = new File(scratchDir, "qnstopp.nml");
          URL qnstoppModelUrl = (URL) qnstoppContext.getModelInput();
          File qnstoppModelFile = new File(scratchDir, "modelInfo.txt");
          GenericUtil.download(qnstoppInputUrl, qnstoppInputFile);
          GenericUtil.download(qnstoppModelUrl, qnstoppModelFile);
```

```
            /* Get system properties. */
            String qnstoppExec = null;
            if (GenericUtil.isLinux()) qnstoppExec =
            getProperty("provider.exec.qnstopp.linux64").trim();
            if (GenericUtil.isMac()) qnstoppExec =
            getProperty("provider.exec.qnstopp.mac64").trim();

            /* Create script to run QNSTOPP. */
            File nativeHome = new File(qnstoppExec).getParentFile();
            Vector<String> script = new Vector<String>();
            script.add("#!/bin/bash");
            script.add("cd " + scratchDir.getAbsolutePath());
            script.add("cp " + new File(qnstoppExec).getAbsolutePath() + " .");
            script.add("cp " + nativeHome + "/qnstoppObjFunc.class .");
            script.add("cp " + nativeHome + "/qnstopp .");
            script.add("./" + new File(qnstoppExec).getName() + " &>
            qnstoppStdOutAndError.txt");

            /* Run shell script. */
            File scriptFile = new File(scratchDir, "script.sh");
            GenericUtil.setFileContents(scriptFile, script);
            scriptFile.setExecutable(true);
            Process p = Runtime.getRuntime().exec(new String[]"sh", "-c",
            scriptFile.getAbsolutePath());
            p.waitFor();

            /* Add output to Context. */
            File qnstoppoutfile1 = new File(scratchDir, "QNSTOPPOutput.txt");
            URL qnstoppOutputFile = this.getScratchURL(qnstoppoutfile1);
            qnstoppContext.setOutput(qnstoppOutputFile);
            logger.info("QNSTOPPContext = " + qnstoppContext);
        } catch (Exception e) {
            StringBuilder reason = new StringBuilder();
            reason.append("error: qnstoppProviderImpl caught exception =
            ").append(e.getClass().getName())
                   .append("scratchDir = ").append(scratchDir)
                   .append("scratchUrl = ").append(scratchUrl);
            context.reportException(reason.toString(), e,
            getProviderInfo("execute"));
            logger.log(Level.SEVERE, reason.toString(), e);
            return context;
        }
        logger.info("Returning from service execute; Output context =
        "+context);
        return context;
    }
}
```

The following method `QnstoppRequestor.java` is a requestor that creates exertions and submits them to the grid in order to utilize services provided by the corresponding analysis provider.

```
/*
  Distribution Statement:
  This computer software has been developed under sponsorship of the United
  States Air Force Research Lab. Any further distribution or use by anyone
```

```
  or any data contained therein, unless otherwise specifically provided for,
  is prohibited without the written approval of AFRL/RQVC-MSTC, 2210 8th
  Street Bldg 146, Room 218, WPAFB, OH  45433.

  Disclaimer:
  This material was prepared as an account of work sponsored by an agency of
  the United States Government. Neither the United States Government nor the
  United States Air Force, nor any of their employees, makes any warranty,
  express or implied, or assumes any legal liability or responsibility for
  the accuracy, completeness, or usefulness of any information, apparatus,
  product, or process disclosed, or represents that its use would not infringe
  privately owned rights.
*/
package mil.afrl.mstc.open.qnstopp.requestor;

import mil.afrl.mstc.open.qnstopp.Qnstopp;
import mil.afrl.mstc.open.data.DataService;
import mil.afrl.mstc.open.provider.ProviderStrategy;
import mil.afrl.mstc.open.requestor.EngineeringRequestor;
import mil.afrl.mstc.open.qnstopp.Qnstopp;
import mil.afrl.mstc.open.qnstopp.QnstoppContext;
import sorcer.core.requestor.SorcerRequestor;
import mil.afrl.mstc.open.task.TaskFactory;
import sorcer.core.context.ServiceContext;
import sorcer.core.exertion.NetTask;
import sorcer.core.signature.NetSignature;
import sorcer.service.Context;
import sorcer.service.Exertion;
import sorcer.util.Sorcer;
import sorcer.util.GenericUtil;
import org.sorcer.core.requestor.SorcerTester;
import java.io.File;
import java.util.Properties;
import java.util.logging.Logger;
import java.net.URL;
import java.util.Vector;

/*
  A requestor that uses the following provider:
  {mil.afrl.mstc.open.qnstopp.Qnstopp}.
  @author MSTC Engineering Project Generator
*/
public class QnstoppRequestor extends SorcerRequestor {
   private static Logger logger =
   Logger.getLogger(QnstoppRequestor.class.getName());
   private File dataDir = new File(System.getProperty("data.dir"));
   private DataService dataService; private Properties properties;
   private EngineeringRequestor engineeringRequestor = new
   EngineeringRequestor();

   /*
     Run the requestor.
     @param args: Arguments to process.  @throws Exception if there are
     problems running the requestor.
   */
   public void run(String... args) throws Exception {
```

```
File engHome = new File(System.getProperty("eng.home"));
File dataDir = new File(engHome, "data");
File tempDir = new File(dataDir, "temp");

/* Get input file for QNSTOPP and model input file. */
File qnstoppInputFile = new
File(engHome,"/optimization/qnstopp/qnstopp-req/data/qnstopp.nml");
File modelInputFile = new
File(engHome,"/optimization/qnstopp/qnstopp-req/data/modelInfo.txt");

/* Edit modeInputFile to adjust model name for user. */
Vector<String> modelInputFileContents =
GenericUtil.getFileContents(modelInputFile);
GenericUtil.printVect(modelInputFileContents);
modelInputFileContents.setElementAt(Sorcer.getActualName
(modelInputFileContents.elementAt(0).trim()), 0);
GenericUtil.printVect(modelInputFileContents);
File modelInputFileTemp = new File(tempDir, "modelInfo.txt");
GenericUtil.setFileContents(modelInputFileTemp,
modelInputFileContents);

/* Copy input files to the scratch directory. */
URL qnstoppInputUrl =
SorcerTester.copyFileToScratchAndGetUrl(qnstoppInputFile, dataDir);
URL modelInputUrl =
SorcerTester.copyFileToScratchAndGetUrl(modelInputFileTemp, dataDir);

/* Add input to Context. */
QnstoppContext context = new QnstoppContext("QNSTOPPContext");
context.setQNSTOPPInput(qnstoppInputUrl);
context.setModelInput(modelInputUrl);

/* Construct execute method. */
String providerName = Sorcer.getActualName("Engineering-Qnstopp");
String serviceName = "execute";
NetSignature methodEN = new NetSignature(serviceName,
       Qnstopp.class,
       providerName);

/* Construct Task. */
NetTask qnstoppTask = new NetTask("run execute", "Task to run
QNSTOPP", methodEN);

/* Put the context into the task. */
qnstoppTask.setContext(context);
logger.info("qnstoppTask = " + qnstoppTask);

/* Exert collaboration. */
Exertion result = qnstoppTask.exert();

logger.info("Returned Task after exert: " + result);
logger.info("Context output =
"+((QnstoppContext)result.getContext()).getOutput());

/* Read output from Context. */
URL qnstoppOutputUrl = (URL) ((QnstoppContext)
```

```
        result.getContext()).getOutput();
        Vector<String> qnstoppOutputFileContents =
        GenericUtil.getFileContents(qnstoppOutputUrl);
        GenericUtil.printVect(qnstoppOutputFileContents);
    }
}
```

Appendix E: Implementation of Model Provider for Objective Function Evaluation

The following method `ModelProviderObjFunc.java` contains the dependent and independent vars that define a specific optimization problem. The fundamentals of a model provider are discussed in Chapter 3.3.

```
/*
  Distribution Statement:
  This computer software has been developed under sponsorship of the United
  States Air Force Research Lab. Any further distribution or use by anyone
  or any data contained therein, unless otherwise specifically provided for,
  is prohibited without the written approval of AFRL/RQVC-MSTC, 2210 8th
  Street Bldg 146, Room 218, WPAFB, OH  45433.

  Disclaimer:
  This material was prepared as an account of work sponsored by an agency of
  the United States Government. Neither the United States Government nor the
  United States Air Force, nor any of their employees, makes any warranty,
  express or implied, or assumes any legal liability or responsibility for
  the accuracy, completeness, or usefulness of any information, apparatus,
  product, or process disclosed, or represents that its use would not infringe
  privately owned rights.
 */
package mil.afrl.mstc.products.OptModel;

import mil.afrl.mstc.open.panelopt.PaneloptContext;
import org.sorcer.core.requestor.SorcerTester;
import sorcer.core.context.model.PoolStrategy;
import sorcer.core.exertion.NetTask;
import sorcer.core.signature.NetSignature;
import sorcer.core.signature.ObjectSignature;
import sorcer.modeling.core.context.model.opti.OptimizationModel;
import sorcer.modeling.vfe.Filter;
import sorcer.modeling.vfe.Var;
import sorcer.modeling.vfe.evaluator.ExertionEvaluator;
import sorcer.modeling.vfe.evaluator.MethodEvaluator;
import sorcer.modeling.vfe.filter.*;
import sorcer.modeling.vfe.util.VarList;
import sorcer.service.Context;
import sorcer.service.ContextException;
import sorcer.service.Getter;
import sorcer.service.SignatureException;
import sorcer.service.modeling.Variability;
import sorcer.util.Sorcer;
import sorcer.service.Strategy;
import sorcer.service.Strategy.Access;
import java.io.File;
import java.net.URL;
import java.util.Properties;
import java.util.logging.Logger;

public class PanelOptModel {
   protected static final Logger logger =
```

```
Logger.getLogger(PanelOptModel.class.getName());
public static Properties props;

/* Load provider properties. */
static {
   if (props == null) {
Sorcer.loadPropertiesNoException(System.getProperty("provider.properties",
"provider.properties"));
   }
}

public static OptimizationModel getModel() {
   OptimizationModel model = null;
   try {
      /* Initialize the design variables. */
Double[] dvArray = {0.0, 1.1, 2.2, 3.3, 4.4, 5.5, 6.6, 7.7,
8.8, 9.9, 10.10, 11.11, 12.12, 13.13, 14.14, 15.15, 16.16, 17.17,
18.18, 19.19,20.20, 21.21, 22.22, 23.23, 24.24};
      VarList designVars = new VarList();

      /* Initialize individual vars. */
      Var<Double> x1 = getDesignVar(0, dvArray, designVars);
      Var<Double> x2 = getDesignVar(1, dvArray, designVars);
      Var<Double> x3 = getDesignVar(2, dvArray, designVars);
      Var<Double> x4 = getDesignVar(3, dvArray, designVars);
      Var<Double> x5 = getDesignVar(4, dvArray, designVars);
      Var<Double> x6 = getDesignVar(5, dvArray, designVars);
      Var<Double> x7 = getDesignVar(6, dvArray, designVars);
      Var<Double> x8 = getDesignVar(7, dvArray, designVars);
      Var<Double> x9 = getDesignVar(8, dvArray, designVars);
      Var<Double> x10 = getDesignVar(9, dvArray, designVars);
      Var<Double> x11 = getDesignVar(10, dvArray, designVars);
      Var<Double> x12 = getDesignVar(11, dvArray, designVars);
      Var<Double> x13 = getDesignVar(12, dvArray, designVars);
      Var<Double> x14 = getDesignVar(13, dvArray, designVars);
      Var<Double> x15 = getDesignVar(14, dvArray, designVars);
      Var<Double> x16 = getDesignVar(15, dvArray, designVars);
      Var<Double> x17 = getDesignVar(16, dvArray, designVars);
      Var<Double> x18 = getDesignVar(17, dvArray, designVars);
      Var<Double> x19 = getDesignVar(18, dvArray, designVars);
      Var<Double> x20 = getDesignVar(19, dvArray, designVars);
      Var<Double> x21 = getDesignVar(20, dvArray, designVars);
      Var<Double> x22 = getDesignVar(21, dvArray, designVars);
      Var<Double> x23 = getDesignVar(22, dvArray, designVars);
      Var<Double> x24 = getDesignVar(23, dvArray, designVars);
      Var<Double> x25 = getDesignVar(24, dvArray, designVars);

      File dataDir = new File(System.getProperty("data.dir"));
      File engHome = new File(System.getProperty("eng.home"));
      /* Copy input file to the scratch directory. */
      File panelOptInputFile = new
      File(engHome,"/products/vtex1-mdl/data/stiffened_plate_input_data.txt");
      URL panelOptInputUrl =
      SorcerTester.copyFileToScratchAndGetUrl(panelOptInputFile,
      dataDir);
```

```
        /* Add input to Context. */
        PaneloptContext ctx = new PaneloptContext();
        ctx.setPanelOptInput(panelOptInputUrl);
        ctx.dvArray = dvArray;

        /* Option to use space or catalog.*/
        boolean useSpace = true;
        boolean useIntra = false;

        /* Invocation to ExertionEvaluator.*/
        ExertionEvaluator yEval = getExertionEvaluator(ctx, useSpace, useIntra);
        yEval.addArgs(designVars);

        /* Initialize the response variables. */
        VarList dependentVars = new VarList();
        Var<Double> y1 = getOutputVar(0, yEval, dependentVars);
        Var<Double> y2 = getOutputVar(1, yEval, dependentVars);
        Var<Double> y3 = getOutputVar(2, yEval, dependentVars);
        Var<Double> y4 = getOutputVar(3, yEval, dependentVars);
        Var<Double> y5 = getOutputVar(4, yEval, dependentVars);

        /* Construct model with dependent and independent vars. */
model = constructModel("PanelOptModel", designVars, dependentVars,
        new VarList(), new VarList());

/*
          The information below is used when a table is passed to the
          model;  if no explicit strategy+builder is passed in the context
          that executes a table, the builder is pulled from the strategy
          which was published with the model.
         */
        PoolStrategy strategy = new PoolStrategy();
        strategy.setBuilder(new ObjectSignature("getModel",
        PanelOptModel.class));
        model.setStrategy(strategy);

    } catch (Exception e) {
        logger.severe(e.toString());
        e.printStackTrace();
    }
    return model;
}

private static OptimizationModel constructModel(String modelName
           , VarList designVarList, VarList dependentVarList , VarList
           objVarList, VarList conVarList) throws ContextException {
    /* Initialize model */
    for (Var<?> v : designVarList) {
        v.setType(Variability.Type.INPUT);
        v.addKind(Variability.Type.DESIGN);
    }
    for (Var<?> v : dependentVarList) {
        v.setType(Variability.Type.OUTPUT);
    }

    /* Construct model. */
```

```
        OptimizationModel omodel = new OptimizationModel(modelName);
        omodel.putConstantVars(new VarList());
        omodel.putLinkedVars(new VarList());
        omodel.putConstraintVars(new VarList());
        omodel.putWatchableVars(new VarList());
        omodel.putObjectiveVars(new VarList());
        omodel.putInvariantVars(new VarList());
        omodel.putInputVars(designVarList);
        omodel.putOutputVars(dependentVarList);
        omodel.putObjectiveVars(objVarList);
        omodel.putConstraintVars(conVarList);
        return omodel;
    }

    protected static ExertionEvaluator getExertionEvaluator(Context<?> context,
                            boolean useSpace,
                            boolean useIntra) throws SignatureException {
        Task task = null;
        String serviceOp = "execute";
        if (useIntra) {
            task = task(sig(serviceOp, WingoptProviderImpl.class), context);
        } else {
            NetSignature method = new NetSignature(serviceOp,
            mil.afrl.mstc.open.wingopt.Wingopt.class,
            Sorcer.getActualName("Engineering-Wingopt"));
            task = new NetTask("Task name: execute", method);
            task.setContext(context);
            /* Logic for intra. */
            task.getControlContext().setAccessType(Strategy.Access.PUSH);
            /* Logic for space. */
            if (useSpace) task.getControlContext().setAccessType(Strategy.Access.PULL);
        }
        return new ExertionEvaluator(task);
    }

    private static Var<Double> getDesignVar(int index, Double[] dvArray,
    VarList designVars) {
        Var<Double> var = new Var("x" + index, dvArray[index]);
        var.setSetter(new ArraySetter(dvArray, index));
        designVars.add(var);
        return var;
    }

    private static Var<Double> getOutputVar(int index, ExertionEvaluator eval,
    VarList dependentVars) {
        Filter contextGetter = new ContextGetter();
        Filter outputFilter = new ObjectFieldFilter("outputArray");
        Var<Double> y = new Var<Double>("y" + index);
        y.setEvaluator(eval);
        y.setFilter(new Filter(contextGetter, outputFilter, new
        ArrayFilter(index)));
        dependentVars.add(y);
        return y;
    }
}
```

Appendix F: EBF3PanelOpt as a SORCER Service

The following method `PaneloptProviderImpl.java` ilustrates the implementation of the EBF3PanelOpt framework as an analysis provider. The method illustrates the use of `GenericUtil` for script robustness.

```
/*
  Distribution Statement:
  This computer software has been developed under sponsorship of the United
  States Air Force Research Lab. Any further distribution or use by anyone
  or any data contained therein, unless otherwise specifically provided for,
  is prohibited without the written approval of AFRL/RQVC-MSTC, 2210 8th
  Street Bldg 146, Room 218, WPAFB, OH  45433.

  Disclaimer:
  This material was prepared as an account of work sponsored by an agency of
  the United States Government. Neither the United States Government nor the
  United States Air Force, nor any of their employees, makes any warranty,
  express or implied, or assumes any legal liability or responsibility for
  the accuracy, completeness, or usefulness of any information, apparatus,
  product, or process disclosed, or represents that its use would not infringe
  privately owned rights.
*/
package mil.afrl.mstc.open.panelopt.provider;
import com.sun.jini.start.LifeCycle;
import mil.afrl.mstc.open.panelopt.Panelopt;
import mil.afrl.mstc.open.panelopt.PaneloptContext;
import mil.afrl.mstc.open.core.provider.EngineeringProvider;
import net.jini.config.ConfigurationException;
import sorcer.service.Context;
import sorcer.util.GenericUtil;
import java.io.File;
import java.io.IOException;
import java.net.URL;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.Vector;
/*
  Implementation of the {mil.afrl.mstc.open.panelopt.Panelopt} interface.
  @author MSTC Engineering Project Generator
 */
public class PaneloptProviderImpl extends EngineeringProvider implements
Panelopt {
    private final static Logger logger =
    Logger.getLogger(PaneloptProviderImpl.class.getName());

    /*
      Default constructor initializes the provider with the properties file
      loaded as a resource.
```

```
  @throws Exception if initialization fails.
 */
public PaneloptProviderImpl() throws Exception {
   this("panelopt/provider.properties");
}

/*
  Create the provider using a properties file
  @param propFile The properties file, must not be {@code null}.
  @throws Exception If initialization fails.
 */
public PaneloptProviderImpl(final String propFile) throws Exception {
   super(propFile);
   initialize();
}

/*
  Create the provider if called from the River {@link
  com.sun.jini.start.ServiceStarter} approach.
  @param args: Parameters that point to configuration file and possible
  overrides. Never {@code null}.
  @param lifeCycle: The lifecycle container to be notified if the
  provider terminates.
  @throws Exception if initialization fails.
 */
public PaneloptProviderImpl(final String[] args, final LifeCycle lifeCycle)
throws Exception {
   super(args, lifeCycle);
   initialize();
}

/* Provides provider specific initialization and environment checking */
private void initialize() throws ConfigurationException {
   /* Provides support for JavaSpaces */
   initSpaceSupport();
}

public Context execute(final Context context) {
   long startTime = System.currentTimeMillis();
   logger.info("Running execute");
   File scratchDir = null;
   URL scratchUrl = null;
   String serviceIdString = doThreadMonitor(null);
   try {
      PaneloptContext paneloptContext = (PaneloptContext) context;
      logger.info("Obtaining scratch directory ... ");
      /* Obtain scratch directory. */
      scratchDir = getScratchDir(context, "panelopt_execute_");
      logger.info(String.format("Scratch directory = [%s]",
      scratchDir.getAbsolutePath()));
      scratchUrl = getScratchURL(scratchDir);
      logger.info(String.format("Scratch URL = [%s]", scratchUrl));

      /* Get the input file and write it out locally. */
      URL paneloptInputURL = (URL) paneloptContext.getInput0();
      File paneloptInputFile = new File(scratchDir,
```

```
    "stiffened_plate_input_data.txt");
    GenericUtil.download(paneloptInputURL, paneloptInputFile);

    /* Write design points to a file. */
    File paneloptDesignVarsFile = new File(scratchDir, "dvar.vef");
    if (!paneloptDesignVarsFile.exists()) {
       paneloptDesignVarsFile.createNewFile();
    }

    /* Get system property. */
    String paneloptExecProp = null;
    if (GenericUtil.isLinux()) paneloptExecProp =
    getProperty("provider.exec.panelopt.linux64").trim();
    if (GenericUtil.isMac()) paneloptExecProp =
    getProperty("provider.exec.panelopt.mac64").trim();
    File panelOptExec = new File(paneloptExecProp);
    File panelOptExecHome = panelOptExec.getParentFile();

    /* Get scratch directory. */
    Vector<String> script = new Vector<String>();
    script.add("#!/bin/bash");
    script.add("cd " + scratchDir.getAbsolutePath());
    script.add("cp " + panelOptExecHome.getAbsolutePath() + "/* .");
    script.add("./" + panelOptExec.getName());

    /* Get scratch directory. */
    File panelOptScriptFile = new File(scratchDir, "doIt.sh");
    File panelOptLogFile = new File(scratchDir,
    panelOptScriptFile.getName() + "Log.txt");

    /* Run python script */
    int exitStatus = 1;
    long lStartTime = System.currentTimeMillis();
    exitStatus = GenericUtil.runShellScript(panelOptScriptFile,
           script, panelOptLogFile, 0, false, true, true);
    logger.info("finished executing script via systemCall...exitValue =
    " + exitStatus);

    /* Get objective function values from file */
    File panelOptoutfile1 = new File(scratchDir, "resp.vef");
    Vector<String> outputFileContents =
    GenericUtil.getFileContents(panelOptoutfile1);
    Double[] outputArray = new Double[outputFileContents.size()];
    int i = 0;
    for (String value : outputFileContents) {
       outputArray[i++] = new Double(value);
    }
    logger.info("outputArray = " + outputArray);
    paneloptContext.outputArray = outputArray;
    logger.info("panelOptContext = " + paneloptContext);
} catch (Exception e) {
    StringBuilder reason = new StringBuilder();
    reason.append("error: PaneloptProviderImpl caught exception =
    ").append(e.getClass().getName())
       .append("scratchDir = ").append(scratchDir) .append("scratchUrl
       = ").append(scratchUrl);
```

```
            context.reportException(reason.toString(), e,
            getProviderInfo("execute"));
            logger.log(Level.SEVERE, reason.toString(), e);
            return context;
        } finally {
            doThreadMonitor(serviceIdString);
            doTimeKeeping((System.currentTimeMillis() - startTime) / 1000);
        }
        logger.info("Returning from service execute; Output context =
        "+context); return context;
        return context;
    }
    private void setFileContents(File file, Object[] objA) throws IOException
    {
        Vector<String> vect = new Vector<String>();
        for (Object o : objA) {
            vect.add(o.toString());
        }
        GenericUtil.setFileContents(file, vect);
    }
}
```

The following method `PaneloptProviderImpl.java` ilustrates the implementation of the EBF3PanelOpt framework as an analysis provider. This implementation illustrates the use of function calls to the corresponding python script and does not incorporate script robustness.

```
/*
  Distribution Statement:
  This computer software has been developed under sponsorship of the United
  States Air Force Research Lab. Any further distribution or use by anyone
  or any data contained therein, unless otherwise specifically provided for,
  is prohibited without the written approval of AFRL/RQVC-MSTC, 2210 8th
  Street Bldg 146, Room 218, WPAFB, OH  45433.

  Disclaimer:
  This material was prepared as an account of work sponsored by an agency of
  the United States Government. Neither the United States Government nor the
  United States Air Force, nor any of their employees, makes any warranty,
  express or implied, or assumes any legal liability or responsibility for
  the accuracy, completeness, or usefulness of any information, apparatus,
  product, or process disclosed, or represents that its use would not infringe
  privately owned rights.
*/

package mil.afrl.mstc.open.panelopt.provider;
import com.sun.jini.start.LifeCycle;
import mil.afrl.mstc.open.panelopt.Panelopt;
import mil.afrl.mstc.open.panelopt.PaneloptContext;
import mil.afrl.mstc.open.core.provider.EngineeringProvider;
import net.jini.config.ConfigurationException;
import sorcer.service.Context;
import sorcer.util.GenericUtil;
import java.io.File;
import java.io.IOException;
import java.net.URL;
```

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.nio.charset.Charset;
import java.nio.file.Files;
import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.Vector;

/*
  Implementation of the {mil.afrl.mstc.open.panelopt.Panelopt} interface.
  @author MSTC Engineering Project Generator
 */
public class PaneloptProviderImpl extends EngineeringProvider implements
Panelopt {
   private final static Logger logger =
   Logger.getLogger(PaneloptProviderImpl.class.getName());

   /*
     Default constructor initializes the provider with the properties file
     loaded as a resource.
     @throws Exception if initialization fails.
    */
   public PaneloptProviderImpl() throws Exception {
      this("panelopt/provider.properties");
   }

   /*
     Create the provider using a properties file
     @param propFile The properties file, must not be {@code null}.
     @throws Exception If initialization fails.
    */
   public PaneloptProviderImpl(final String propFile) throws Exception {
      super(propFile);
      initialize();
   }

   /*
     Create the provider if called from the River {@link
     com.sun.jini.start.ServiceStarter} approach.
     @param args: Parameters that point to configuration file and possible
     overrides. Never {@code null}.
     @param lifeCycle: The lifecycle container to be notified if the
     provider terminates.
     @throws Exception if initialization fails.
    */
   public PaneloptProviderImpl(final String[] args, final LifeCycle lifeCycle)
   throws Exception {
      super(args, lifeCycle);
      initialize();
   }

   /* Provides provider specific initialization and environment checking */
   private void initialize() throws ConfigurationException {
      initSpaceSupport();
```

```
    if (GenericUtil.isLinux()) panelOptExec = new
    File(getProperty("provider.exec.panelopt.linux64").trim());
    if (GenericUtil.isMac()) panelOptExec = new
    File(getProperty("provider.exec.panelopt.mac64").trim());
    panelOptNativeDir = panelOptExec.getParentFile();
    panelOptNativeDir = panelOptExec.getParentFile();
    logger.info("panelOptNativeDir = " + panelOptNativeDir);
}
private File panelOptExec = null;
private File panelOptNativeDir = null;
private static int threadsRunning = 0;
public Context execute(final Context context) {
    long startTime = System.currentTimeMillis();
    logger.info("************* running execute *****************");
    logger.info("incremented threadsRunning = " + ++threadsRunning);
    File scratchDir = null;
    URL scratchUrl = null;
    PaneloptContext paneloptContext = (PaneloptContext) context;
    try {
        /* Obtain scratch directory. */
        scratchDir = getScratchDir(context, "panelopt_execute_");
        logger.info("scratchDir = " + scratchDir.getAbsolutePath());
        scratchUrl = getScratchURL(scratchDir);
        logger.info("scratchUrl = " + scratchUrl);

        /* Write design points to a file. */
        File paneloptDesignVarsFile = new File(scratchDir, "dvar.vef");
        setFileContents(paneloptDesignVarsFile, paneloptContext.dvArray);

        /* Create script file to run EBF3PanelOpt */
        File panelOptScriptFile = new File(scratchDir, "doIt.sh");
        File panelOptLogFile = new File(scratchDir,
        panelOptScriptFile.getName() + "Log.txt");
        panelOptScriptFile.createNewFile();
        panelOptScriptFile.setExecutable(true, false);
        FileWriter fw = new FileWriter(panelOptScriptFile, true);
        fw.write("#!/bin/bash\ n");
        fw.write("cd " + scratchDir.getAbsolutePath() + "\ n");
        fw.write("cp " + panelOptNativeDir.getAbsolutePath() + "/* . \ n");
        fw.write("./" + panelOptExec.getName() + " &> " + "
        stdOutAndError.txt \ n"); fw.close();
        fw.close();
        logger.info("calling: sh -c " +
        panelOptScriptFile.getAbsolutePath());
        long sysCallStartTime = System.currentTimeMillis();

        /* Execute Script */
        Process p = Runtime.getRuntime().exec(new String[]"sh", "-c",
        panelOptScriptFile.getAbsolutePath());
        int exitValue = p.waitFor();
        long sysCallElapsedTimeMilliSeconds = ((System.currentTimeMillis()
        - sysCallStartTime));
        paneloptContext.sysCallElapsedTimeMilliSeconds =
        sysCallElapsedTimeMilliSeconds;
        logger.info("sysCallElapsedTimeMilliSeconds [ms] = " +
        sysCallElapsedTimeMilliSeconds);
```

```
            /* Get objective function values from file */
            File panelOptoutfile1 = new File(scratchDir, "resp.vef");
            Vector<String> outputFileContents =
            GenericUtil.getFileContents(panelOptoutfile1);
            Double[] outputArray = new Double[outputFileContents.size()];
            int i = 0;
            for (String value : outputFileContents) {
               outputArray[i++] = new Double(value);
            }
            paneloptContext.outputArray = outputArray;
         } catch (Throwable e) {
            StringBuilder reason = new StringBuilder();
            reason.append("error: PaneloptProviderImpl caught exception =
            ").append(e.getClass().getName())
               .append("scratchDir = ").append(scratchDir) .append("scratchUrl
               = ").append(scratchUrl);
            context.reportException(reason.toString(), e,
            getProviderInfo("execute"));
            logger.log(Level.SEVERE, reason.toString(), e);
            return context;
         }
         long stopTime = System.currentTimeMillis();
         long serviceOpElapsedTimeMilliSeconds = stopTime - startTime;
         logger.info("serviceOpElapsedTimeMilliSeconds = " +
         serviceOpElapsedTimeMilliSeconds);
         paneloptContext.serviceOpElapsedTimeMilliSeconds =
         serviceOpElapsedTimeMilliSeconds;
         logger.info("deccremented threadsRunning = " +--threadsRunning);
         return paneloptContext;
      }
      private void setFileContents(File file, Object[] objA) throws IOException {
         Vector<String> vect = new Vector<String>();
         for (Object o : objA)
            vect.add(o.toString());
         }
         GenericUtil.setFileContents(file, vect);
      }
}
```

The following method `PaneloptRequestor.java` is a requestor that creates exertions and submits them to the grid in order to utilize services provided by the corresponding analysis provider.

```
/*
  Distribution Statement:
  This computer software has been developed under sponsorship of the United
  States Air Force Research Lab. Any further distribution or use by anyone
  or any data contained therein, unless otherwise specifically provided for,
  is prohibited without the written approval of AFRL/RQVC-MSTC, 2210 8th
  Street Bldg 146, Room 218, WPAFB, OH  45433.

  Disclaimer:
  This material was prepared as an account of work sponsored by an agency of
  the United States Government. Neither the United States Government nor the
  United States Air Force, nor any of their employees, makes any warranty,
```

```
  express or implied, or assumes any legal liability or responsibility for
  the accuracy, completeness, or usefulness of any information, apparatus,
  product, or process disclosed, or represents that its use would not infringe
  privately owned rights.
*/
package mil.afrl.mstc.open.panelopt.requestor;
import mil.afrl.mstc.open.panelopt.Panelopt;
import mil.afrl.mstc.open.panelopt.PaneloptContext;
import mil.afrl.mstc.open.data.DataService;
import mil.afrl.mstc.open.requestor.EngineeringRequestor;
import org.sorcer.core.requestor.SorcerTester;
import sorcer.core.exertion.NetTask;
import sorcer.core.requestor.SorcerRequestor;
import sorcer.core.signature.NetSignature;
import sorcer.service.Exertion;
import sorcer.util.GenericUtil;
import sorcer.util.Sorcer;
import java.io.*;
import java.lang.*;
import java.net.URL;
import java.util.Properties;
import java.util.Vector;
import java.util.logging.Logger;
/*
  A requestor that uses the following provider:
  {mil.afrl.mstc.open.panelopt.Panelopt}.
  @author MSTC Engineering Project Generator
 */
public class PaneloptRequestor extends SorcerRequestor {
    private static Logger logger =
    Logger.getLogger(PaneloptRequestor.class.getName());
    private File dataDir = new File(System.getProperty("data.dir"));
    private DataService dataService;
    private Properties properties;
    private EngineeringRequestor engineeringRequestor = new
    EngineeringRequestor();
    /*
      Run the requestor.
      @param args Arguments to process.  @throws Exception if there are
      problems running the requestor.
     */
    public void run(String... args) throws Exception {
        File engHome = new File(System.getProperty("eng.home"));
        File dataDir = new File(engHome, "data");
        File tempDir = new File(dataDir, "temp");

        /* Get input files required by the EBF3PanelOpt framework. */
        File panelOptInputFile = new
        File(engHome,"/products/panelopt/panelopt-req/data/stiffened_plate_input_data.txt");
        File panelOptDesVarsInputFile = new
        File(engHome,"/products/panelopt/panelopt-req/data/dvar.vef");

        /* Copy input files to the scratch directory. */
        URL panelOptInputUrl =
        SorcerTester.copyFileToScratchAndGetUrl(panelOptInputFile, dataDir);
        URL panelOptDesVarsInputUrl =
```

```
        SorcerTester.copyFileToScratchAndGetUrl(panelOptDesVarsInputFile,
        dataDir);

        /* Add input to Context. */
        PaneloptContext("PaneloptContext");
        context.setPanelOptInput(panelOptInputUrl);
        Vector<String> desVarStrings =
        GenericUtil.getFileContents(panelOptDesVarsInputUrl);
        Double[] dvArray = new Double[desVarStrings.size()];
        int i = 0;
        for (String dvs : desVarStrings) {
           dvArray[i++] = new Double(dvs);
        }
        context.dvArray = dvArray;

        /* Construct execute method. */
        String providerName = Sorcer.getActualName("Engineering-Panelopt");
        String serviceName = "execute";
        NetSignature methodEN = new NetSignature(serviceName,
               Panelopt.class,
               providerName);

        /* Construct Task. */
        NetTask panelOptTask = new NetTask("run execute", "Task to run panel
        opt", methodEN);

        /* Put the context into the task. */
        logger.info("panelOptTask = " + panelOptTask);
        Exertion result = panelOptTask.exert();
        logger.info("Returned Task after exert: " + result);
        logger.info("outputArray = "+
        GenericUtil.arrayToString(((PaneloptContext)result.getContext()).outputArray));
        System.out.println("dvArray = "+
        GenericUtil.arrayToString(((PaneloptContext)result.getContext()).dvArray));
        System.out.println("outputArray = " +
        GenericUtil.arrayToString(((PaneloptContext)
        result.getContext()).outputArray));
    }
}
```

Appendix G: Implementation of Model Client for Objective Function Evaluation

As discussed in Chapter 3.3., 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. The following method `evaluate` contains the model client implementation and interacts with the published model provider via a single model query.

```
/*
  Distribution Statement:
  This computer software has been developed under sponsorship of the United
  States Air Force Research Lab. Any further distribution or use by anyone
  or any data contained therein, unless otherwise specifically provided for,
  is prohibited without the written approval of AFRL/RQVC-MSTC, 2210 8th
  Street Bldg 146, Room 218, WPAFB, OH  45433.

  Disclaimer:
  This material was prepared as an account of work sponsored by an agency of
  the United States Government. Neither the United States Government nor the
  United States Air Force, nor any of their employees, makes any warranty,
  express or implied, or assumes any legal liability or responsibility for
  the accuracy, completeness, or usefulness of any information, apparatus,
  product, or process disclosed, or represents that its use would not infringe
  privately owned rights.
 */

import sorcer.modeling.client.ModelClient;
import sorcer.util.GenericUtil;
import sorcer.modeling.client.ClientUtils;
import java.io.File;
import java.lang.Exception;
import java.lang.StringBuilder;
import java.lang.Throwable;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Vector;
import sorcer.modeling.vo.operator;
import sorcer.modeling.core.context.model.explore.ResponseContext;
import sorcer.modeling.service.ParametricModeling;
import sorcer.modeling.vfe.util.VarInfoList;
import sorcer.modeling.vfe.VarInfo;
import sorcer.service.Arg;
import sorcer.service.Task;
import sorcer.core.context.ThrowableTrace;

public class objFunc {
    public static ArrayList<String> xNames = new ArrayList<String>();
    public static String objName = null;
    public static int numberDesignVariables = 0;
    public static VarInfoList objVarInfoList = new VarInfoList();
    public static VarInfoList designVarInfoList = new VarInfoList();
    public static String modelName = null;
    public static boolean isTableRun = false;
    public static boolean printMessage = false;
```

```
static {
    ClientUtils.setDoLog(false);
    File modelInfoFile = new File("modelInfo.txt");
    Vector<String> modelInfo = null;

    try {
        String policyProperty = System.getProperty("java.security.policy");
        System.out.println("setting policyProperty in jvm to: " +
        policyProperty);
        ClientUtils.setPolicyAndSecurity(policyProperty);

        /*
          Assumes 'modelInfo.txt' file is present and contains one entry
          per line:
          line 1: model name
          line 2: objective function name
          lines 3-(ndv+2): design variable names.
         */
        modelInfo = GenericUtil.getFileContents(modelInfoFile);
        Iterator<String> it = modelInfo.iterator();

        /* Initialize model client with model name in modelInfo.txt file. */
        modelName = it.next().trim();

        /* Initialize model client with the dependent var's name in modelInfo.txt
           file.
         */
        objName = it.next().trim();
        objVarInfoList.add(new VarInfo(objName));

        StringBuilder sb = new StringBuilder();
        sb.append("objFunc");
        sb.append("modelName = " + modelName);
        sb.append("objName = " + objName + "");
        sb.append("designVars = ");

        /* Add var info to model */
        while (it.hasNext()) {
            String xName = it.next().trim();
            xNames.add(xName);
            numberDesignVariables++;
            sb.append("" + xName);
        }
        printMessage(sb.toString());
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static void printMessage(String msg) {
    if (printMessage) System.out.println(msg);
}

public static double evaluate(double[] designPoints) {
```

```
    /* Get value of dependent var. */
    Double objValue = getObjWithRemoteModelContextQuery(designPoints);
    printMessage("objValue = " + objValue);
    return objValue;
}

private static Double getObjWithRemoteModelContextQuery(double[]
designPoints) {
    Double objValue = null;
    String xVals = "";
    int i = 0;
    designVarInfoList.clear();
    for (String xName : xNames) {
        printMessage("*** ObjjFunc.java: xName = " + xName + ",
        value = " + designPoints[i]);

        /* Add independent var values to model. */
        xVals = xVals + Double.toString(designPoints[i]);
        designVarInfoList.add(operator.varInfo(xName, designPoints[i++]));
    }
    try {
        ResponseContext modelContext = new ResponseContext("ObjFunc
        Context");
        modelContext.setInputVarsInfo(designVarInfoList);
        modelContext.setOutputVarsInfo(objVarInfoList);

        /* Call execute method on model. */
        VarInfoList oil = executeTask(modelContext).getOutputVarsInfo();
        objValue = ((Double) oil.getVarInfo(objName).getValue(new
        Arg[0])).doubleValue();
    } catch (Exception e) {
        e.printStackTrace();
    }
    return objValue;
}

private static ResponseContext executeTask(ResponseContext modelContext)
throws Exception {
    String serviceOp = "evaluate";
    if(isTableRun) serviceOp = "evaluateResponseTable";

    /* Construct Task. */
    Task runModelTask = operator.task("ObjFunc Model Task"
           , sorcer.eo.operator.sig(serviceOp
           , ParametricModeling.class
           , new Arg[]{sorcer.eo.operator.prvName(modelName)})
           , modelContext);
    printMessage("executeTask(): exerting task...task = " + runModelTask);

    /* Exert collaboration. */
    Task taskOut = (Task)runModelTask.exert(new Arg[0]);
    printMessage("executeTask(): done exerting task.");
    if(taskOut.getExceptionCount() > 0) {
        int i = 1;
        Iterator var6 = taskOut.getExceptions().iterator();
        while(var6.hasNext()) {
```

```
            ThrowableTrace trace = (ThrowableTrace)var6.next();
            printMessage("Exception #" + i++ + ": " +
            trace.getStackTrace(trace.getThrowable()));
         }
         throw new Exception("*** error: there were exception(s) executing
         the model: taskOut = " + taskOut + "taskOut.getExceptions() =
         " + taskOut.getExceptions());
      }
      return (ResponseContext)taskOut.getContext();
   }
}
```

The following method `evaluateP` contains the model client implementation that interacts with the published model provider via a table model query.

```
public static double[] evaluateP(double[] designPoints, int P) {
   printMessage("ObjFunc.evaluateP: P = " + P);
   printMessage("ObjFunc.evaluateP: designPoints.length = " +
   designPoints.length);
   return evaluatePModelQuery(designPoints, P);
}

private static double[] evaluatePModelQuery(double[] designPoints,
int P) {
   printMessage("ObjFunc.evaluateP: P = " + P);
   printMessage("ObjFunc.evaluateP: designPoints.length = " +
   designPoints.length);
   int nRuns = (designPoints.length) / P;
   double[] objValuesPrimitives = new double[nRuns];
   printMessage("ObjFunc.evaluateP: nRuns = " + nRuns);

   /* Construct table of design points. */
   try {
      Table inputTable = new Table();
      inputTable.setColumnIdentifiers(xNames);
      int i = 0; int x =0;
      while (x < designPoints.length) {
         i = x;
         ArrayList<Double> xValsList = new ArrayList<Double>();
         for (int j=0; j<xNames.size(); j++) {
            xValsList.add(designPoints[i++]);
         }
         inputTable.addRow(xValsList);
         x = x + P;
      }
      printMessage("P-ObjFunc.evaluateP: inputTable = " +
      inputTable.toString());

      /* Execute table. */
      Table outputTable = executeTable(nRuns, inputTable);
      printMessage("ObjFunc.evaluateP: outputTable = " +
      inputTable.toString());

      /* Pack up objective function vector. */
      List objTableColumn = outputTable.getColumn(objName);
      int k = 0;
      for (Object obj : objTableColumn) {
         objValuesPrimitives[k++] = Double.parseDouble(obj.toString());
         printMessage("ObjFunc.evaluateP: objValuesPrimitives[" +
         (k - 1) + "] = " + objValuesPrimitives[k-1]);
      }
   } catch (Throwable e) {
      e.printStackTrace();
   }
   printMessage("ObjFunc.evaluateP: numberOfParallelFunctionCalls =
   " + ++numberOfParallelFunctionCalls);
   return objValuesPrimitives;
}
```

```
private static Table executeTable(int numberOfSimultaneousParallelRuns
       , Table inputTable) throws Exception {
   printMessage("executeTableRun(): starting...");
   PoolStrategy strategy = new PoolStrategy();
   strategy.setCapacity(1);
   strategy.setLoad(0);
   strategy.setFlow(Flow.PAR);
   printMessage("executeTableRun(): numberOfSimultaneousParallelRuns = "
          + numberOfSimultaneousParallelRuns);
   printMessage("executeTableRun(): flow (always PAR)=  " +
   strategy.getFlow());
   strategy.getPools().add(
          new Pool(ParType.THREAD, numberOfSimultaneousParallelRuns,
                numberOfSimultaneousParallelRuns));
   ParametricContext pc = new ParametricContext(inputTable, strategy);
   pc.setResponseVarsInfo(objVarInfoList);
   printMessage("executeTable(): going to execute parametric table...");
   pc = (ParametricContext) executeTask(pc, true);
   printMessage("executeTable(): done executing parametric table.");
   Table outputTable = pc.getOutTable();
   return outputTable;
}
```