A new approach to engineering design

Christophe Prud'homme, Dimitrios Rovas, and A. T. Patera

Department of Mechanical Engineering, Room 3-243, Massachusetts Institute of Technology, Cambridge, MA, 02139-4307

Abstract. In the following paper, we present two components which have been used together to solve engineering design problems. Firstly, we recall some results on Reduced-Basis Output Bound methods which provide real-time outputs and their associated error estimators for a parametrized mathematical model. Then, we propose an original architecture – called SimRes– for scientific computing which itself comprises several components. Put together, these two components provide a complete solution for certain classes of engineering design problems in terms of numerical methods and software.
keyworks: Blackbox Reduced-Basis output bound methods, engineering design, distributed computing, software design and interfaces, CORBA.

Introduction

We present two components which have been used together to provide a complete solution for some engineering design problems. While these components can vary independently, they fit together quite well. The first one is an ongoing research work on reduced-basis methods; more particularly we will examine in this paper the two stage off-line/on-line blackbox reduced-basis output bound method[1] for the prediction of outputs of coercive partial differential equations with affine parameter dependence. One essential feature of the on-line stage, which allows both components to work well together, is that the computational complexity of this procedure scales only with the dimension of the reduced-basis space and the parametric complexity of the partial differential operator. While the method is efficient, it is also certain thanks to rigorous *a posteriori* error bounds which allow us to retain only the minimal number of modes necessary to achieve the prescribed accuracy in the output of interest. The technique is therefore particularly appropriate for applications such as design and optimization, in which repeated and rapid evaluation of the output is required. In order to illustrate the previous statement, we also developed a software architecture, called SimRes, which takes advantage of the numerical method's features while providing an original interface using the LaTeX typesetting system and PDF [2] as its output. SimRes can be viewed as a client/server architecture whose client side can be somewhat very simple to implement while the server side enjoys a distributed objects architecture over an heterogeneous networked environment using the Common Object Request Broker Architecture (CORBA) [2].

In the first part of this article, we will review briefly certain aspects of the blackbox reduced-basis output bound method, in particular we consider here equilibrium solutions of coercive problems within the context of shape optimization; see also [4] for treatment of noncoercive equilibrium problems and [3] for symmetric eigenvalue problems. Then we will present the architecture of SimRes and its different components. Finally, we will illustrate both the numerical methods and the SimRes technology by a practical example, a 3D fin.

1 Numerical Method

Preliminaries

Let Y be a Hilbert space with an associated inner product $(\cdot,\cdot)_Y$ and an induced norm $\|\cdot\|_Y$. We define our parameter space to be $\mathcal{D} \subset \mathbb{R}$; a point in that space is denoted μ. Our problem is then to find $u \in Y$ such that

$$a(u, v; \mu) = \ell(v),\ \forall v \in Y, \tag{1}$$

[1] For Reduced Basis methods see [1,5,6].

[2] The Portable Document Format from Adobe.

and subsequently the output of interest $s(u) = \ell^0(u)$; $\ell(\cdot)$ and $\ell^0(\cdot)$ are both in Y', the dual space of Y. The bilinear form a is assumed to be continuous; symmetric, $a(w, v; \mu) = a(v, w; \mu), \ \forall w, v \in Y$; and coercive, $a(v, v; \mu) \geq c\|v\|_Y^2 > 0, \ \forall v \in Y, \forall \mu \in \mathcal{D}$, where c is a strictly positive real constant. Associated with the above primal problem we define the dual problem for $\psi \in Y: \ a(v, \psi; \mu) = -\ell^0(v), \ \forall v \in Y$. The need for this problem will become clear in the error estimation discussion.

We next introduce a symmetric positive-definite form $\hat{a}(w, v)$, and define $\lambda_{\hat{a}}^1(\mu)$ to be the minimum eigenvalue of $a(\varphi, v; \mu) = \lambda(\mu)\hat{a}(\varphi, v), \ \forall v \in Y$. A lower bound for this eigenvalue is required by the output bound procedure: we assume that a $g(\mu)$ is known such that

$$a(v, v; \mu) \geq g(\mu)\hat{a}(v, v) > 0, \ \forall v \in Y \text{ and } \forall \mu \in \mathcal{D}. \tag{2}$$

It is also possible to include approximation of $\lambda_{\hat{a}}^1(\mu)$ as part of the reduced basis approximation [4].

Finally, for the blackbox method, we shall assume that, for some finite integer Q, there exists a decomposition of $a(w, v; \mu)$ of the form

$$a(w, v; \mu) = \sum_{q=1}^{Q} \sigma^q(\mu) a^q(w, v), \forall w, v \in Y \text{and } \forall \mu \in \mathcal{D}, \tag{3}$$

where we make no assumptions on the a^q other than continuity and bilinearity.

Reduced-Basis Approximation

We choose $N/2$ points in our parameter space $\mathcal{D}$, and form the sample set $S_N = \{\mu_1, \ldots, \mu_{N/2}\}$. The reduced-basis spaces associated with the primal and dual problems are then given by $W_N^{pr} = \text{span}\{u(\mu_1), \ldots, u(\mu_{N/2})\}$ and $W_N^{du} = \text{span}\{\psi(\mu_1), \ldots, \psi(\mu_{N/2})\}$ respectively; we can now form

$$W_N = \text{span}\{u(\mu_1), \psi(\mu_1), \ldots, u(\mu_{N/2}), \psi(\mu_{N/2})\} \equiv \text{span}\{\zeta_1, \ldots, \zeta_N\}. \tag{4}$$

The space W_N defined this way has good approximation properties both for the primal and the dual problems.

For each new desired $\mu \in \mathcal{D}$, we now apply a standard Galerkin procedure over W_N to obtain $u_N(\mu)$ and $\psi_N(\mu)$ according to $a(u_N(\mu), v; \mu) = \ell(v), \ \forall v \in W_N$, and $a(v, \psi_N(\mu); \mu) = -\ell^0(v), \ \forall v \in W_N$. The output can then be calculated as $s_N(\mu) = \ell^0(u_N(\mu))$.

Bounds Evaluation

We start by defining the residuals associated with the primal and dual reduced-basis approximations, $R^{pr}(v; \mu) = \ell(v) - a(u_N(\mu), v; \mu), \ \forall v \in Y$, and $R^{du}(v; \mu) = -\ell^0(v) - a(v, \psi_N(\mu); \mu), \ \forall v \in Y$, respectively. The Riesz representations $\hat{e}^{pr}(\mu)$ and $\hat{e}^{du}(\mu)$ of the primal and dual residuals can then be defined as $\hat{a}(\hat{e}^{pr}(\mu), v) = R^{pr}(v; \mu), \ \forall v \in Y, \ \hat{a}(\hat{e}^{du}(\mu), v) = R^{du}(v; \mu), \forall v \in Y$.

We then define, as in [3,4],

$$\begin{aligned} \bar{s}_N(\mu) &= s_N(\mu) - \frac{1}{2g(\mu)}\hat{a}(\hat{e}^{pr}(\mu), \hat{e}^{du}(\mu)), \\ \varDelta_N(\mu) &= \frac{1}{2g(\mu)}\hat{a}^{1/2}(\hat{e}^{pr}(\mu), \hat{e}^{pr}(\mu)) \ \hat{a}^{1/2}(\hat{e}^{du}(\mu), \hat{e}^{du}(\mu)), \end{aligned} \tag{5}$$

and compute lower and upper estimators $s_N^{\pm} = \bar{s}_N \pm \varDelta_N$.

It can be shown [3,4] that s_N^+ (respectively s_N^-) will be an upper (respectively lower) bound for s provided that $g(\mu)$ is a lower bound for the eigenvalue $\lambda_{\hat{a}}^1(\mu)$ (or equivalently satisfies ((2))). Note that in the general case, where an $\hat{a}$ and $g(\mu)$ which satisfy ((2)) may not be readily available, the reduced-basis space must be augmented with eigenmodes corresponding to the minimum eigenvalue of the problem $a(\varphi, v; \mu) = \lambda(\mu)\hat{a}(\varphi, v), \ \forall v \in Y$ [4].

Also of interest is the quality of the bounds — how well they approximate the actual error. We measure the quality of the bounds by the effectivity $\eta_N(\mu)$, defined as the ratio of the bound gap $\varDelta_N$ to $|s - \bar{s}_N|$. From the bound result we know that $\eta_N(\mu) \geq 1$. We can further prove [4] that $\eta_N(\mu)$ is bounded independent of N; in practice, $\eta_N(\mu)$ is typically $O(1)$, as desired.

Blackbox Method

The parametric dependence assumed in ((3)) permits us to decouple the computation into two stages: the *off-line* stage, in which (i) the reduced basis is constructed and, (ii) the necessary error-estimation preprocessing is performed; and the *on-line* stage, in which for each new desired value of μ, μ_d, we compute $s_N(\mu_d)$ and the associated bounds. The essential "enabler" is the absence of μ dependence in $\hat{a}$, which allows us to precompute (and later assemble) all the "pieces" of $\hat{e}^{pr}(\mu_d)$, and $\hat{e}^{du}(\mu_d)$ by linear superposition. The details of the blackbox technique follow. For convenience we define $\mathcal{N}$ as the set $\{1,\ldots,N\}$, and $\mathcal{Q}$ as the set $\{1,\ldots,Q\}$.

Off-line **Stage** 1. Calculate $u(\mu_i)$ and $\psi(\mu_i), i=1,\ldots,N/2$, to form W_N as in ((4)).
2. Compute $\underline{A}^q \in \mathbb{R}^{N\times N}$ as $A^q_{i,j} = a^q(\zeta_i,\zeta_j), \forall i,j \in \mathcal{N}^2$ and $\forall q \in \mathcal{Q}$.
3. Solve for $\hat{z}^{0,pr} \in Y$ and $\hat{z}^{0,du} \in Y$ from $\hat{a}(\hat{z}^{0,pr},v) = \ell(v),\ \forall v\in Y$, and $\hat{a}(\hat{z}^{0,du},v) = -\ell^0(v),\ \forall v \in Y$, respectively. Also, compute $\hat{z}^q_j \in Y$ from $\hat{a}(\hat{z}^q_j,v) = -a^q(\zeta_j,v),\ \forall v\in Y, \forall j \in \mathcal{N}$ and $\forall q\in\mathcal{Q}$.
4. Calculate and store $c_0^{pr} = \hat{a}(\hat{z}^{0,pr},\hat{z}^{0,pr})$; $c_0^{du} = \hat{a}(\hat{z}^{0,du},\hat{z}^{0,du})$; $c_0^{pr,du} = \hat{a}(\hat{z}^{0,pr},\hat{z}^{0,du})$; $F^{pr}_{N,j} = \ell(\zeta_j)$ and $F^{du}_{N,j} = \ell^0(\zeta_j),\ \forall j\in\mathcal{N}$; $\Lambda^{q,pr}_j = \hat{a}(\hat{z}^{0,pr},\hat{z}^q_j)$ and $\Lambda^{q,du}_j = \hat{a}(\hat{z}^{0,du},\hat{z}^q_j),\ \forall j \in \mathcal{N}$ and $\forall q\in\mathcal{Q}$; $\Gamma^{pq}_{ij} = \hat{a}(\hat{z}^p_i,\hat{z}^q_j), \forall i,j\in\mathcal{N}^2$ and $\forall p,q \in \mathcal{Q}^2$.
This stage requires $(NQ+N+2)$ Y-linear system solves; $(N^2Q^2+2NQ+3)$ $\hat{a}$-inner products; and $2N$ evaluations of linear functionals.

On-line **Stage** For each new desired design point μ_d we then compute the reduced-basis prediction and error bound based on the quantities computed in the off-line stage.
1. Form $\underline{A}_N = \sum_{q=1}^{Q}\sigma^q(\mu_d)\underline{A}^q$ and solve for $\underline{u}_N \equiv \underline{u}_N(\mu_d)\in\mathbb{R}^N$ and $\underline{\psi}_N \equiv \underline{\psi}_N(\mu_d)\in\mathbb{R}^N$ from $\underline{A}_N\ \underline{u}_N = \underline{F}^{pr}_N$ and $\underline{A}_N\ \underline{\psi}_N = -\underline{F}^{du}_N$, respectively.
2. Evaluate the bound average and bound gap as

$$\begin{aligned}\bar{s}_N = {} & (\underline{F}^{du}_N)^T\underline{u}_N - \\ & \frac{1}{2g(\mu_d)}\Big(\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{p=1}^{Q}\sum_{q=1}^{Q}u_{N,i}\psi_{N,j}\sigma^p(\mu_d)\sigma^q(\mu_d)\Gamma^{pq}_{ij} + \sum_{j=1}^{N}\sum_{q=1}^{Q}\psi_{N,j}\sigma^q(\mu_d)\Lambda^{q,pr}_j + \\ & \qquad \sum_{j=1}^{N}\sum_{q=1}^{Q}u_{N,j}\sigma^q(\mu_d)\Lambda^{q,du}_j + c_0^{pr,du}\Big),\end{aligned}$$

and

$$\begin{aligned}\Delta_N(\mu_d) = {} & \frac{1}{2\ g(\mu_d)}\times \\ & \Big(\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{p=1}^{Q}\sum_{q=1}^{Q}u_{N,i}u_{N,j}\sigma^p(\mu_d)\sigma^q(\mu_d)\Gamma^{pq}_{ij} + 2\sum_{j=1}^{N}\sum_{q=1}^{Q}u_{N,j}\sigma^q(\mu_d)\Lambda^{q,pr}_j + c_0^{pr}\Big)^{\frac{1}{2}}\times \\ & \Big(\sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{p=1}^{Q}\sum_{q=1}^{Q}\psi_{N,i}\psi_{N,j}\sigma^p(\mu_d)\sigma^q(\mu_d)\Gamma^{pq}_{ij} + 2\sum_{j=1}^{N}\sum_{q=1}^{Q}\psi_{N,j}\sigma^q(\mu_d)\Lambda^{q,du}_j + c_0^{du}\Big)^{\frac{1}{2}}.\end{aligned}$$

respectively.
For each μ_d, $O(N^2Q^2+N^3)$ operations are required to obtain the reduced-basis solution and the bounds. Since $\dim(W_N) \ll \dim(Y)$, the cost to compute $s_N(\mu_d)$, $\bar{s}_N(\mu_d)$, and $\Delta_N(\mu_d)$ in the on-line stage will typically be much less than the cost to directly evaluate $u(\mu_d)$ and $s(\mu_d) = \ell^0(u(\mu_d))$ from ((1)).

2 The architecture of SimRes

The purpose of SimRes is two-fold : first, we want to build an online code repository which can be accessed transparently by any client; that is the client doesn't have to know the physical location of the online code; second, we want to provide a simple while powerful interface to the online code repository. This interface should have very few requirements on the user side and should be available on almost all platforms. The figure 1 provides a very basic view of the architecture of SimRes, however it gives its main ingredients and removes the technologies dependencies. In the forthcoming sections we will describe the different part of SimRes. These parts are mostly independent : any dependencies are dealt through interfaces, no data is shared. This is very important because it allows us to develop a system which is scalable and extensible at will.

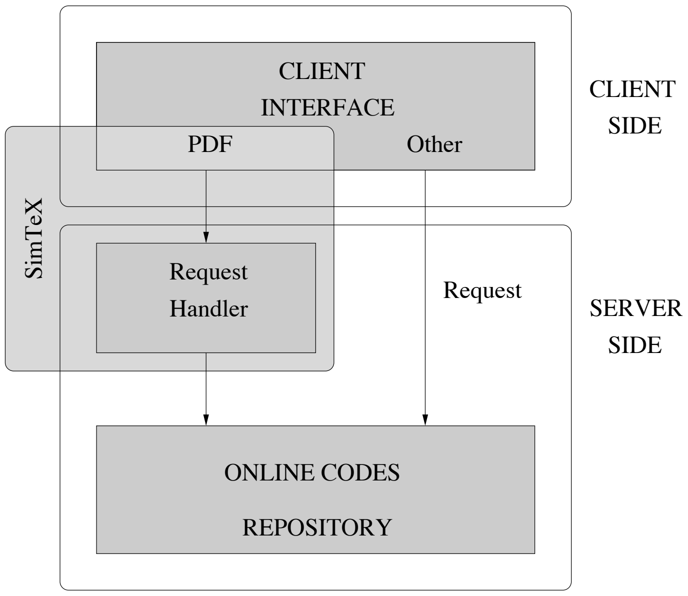

Fig. 1. The architecture of SimRes

Online code repository

In modern programming languages the *object paradigm* is employed to structure computation. This paradigm is very powerful to capture the concepts and abstractions of a wide range of application domains. Scientific computing is one of them. However object orientation might not be the wisest choice in terms of performances or design. In our experience, a successful design is often achieved by considering a variety of paradigms like the meta-computing, the object or generic ones. The C++ language provides them and has been our language of choice for our codes.

Now, classical scientific computing is often done within a single process or uses a parallel computing paradigm like SPMD using a message passing library. Since object orientation has proven to be adequate for the design, implementation and maintenance of large scale applications, it is just one step forward to use this paradigm for distributed computing, that is the objects are distributed over a networked environment and can communicate with each other. That's where stands the Common Object Request Broker Architecture (CORBA) which is a specification of middleware software that allows to ensure the link between the software and the hardware in an heterogeneous networked environment. The figure 2 on the next page illustrates this.

Coming back to our scientific computing concerns, we have seen that the blackbox method split the resolution of the design problem into two steps : an off-line step computes various data which are stored into a database, and an on-line step which computes the corresponding output and its associated error for a given parameter μ. Putting everything together, it is now clear that a component wise architecture over a networked environment is a very good choice for SimRes.

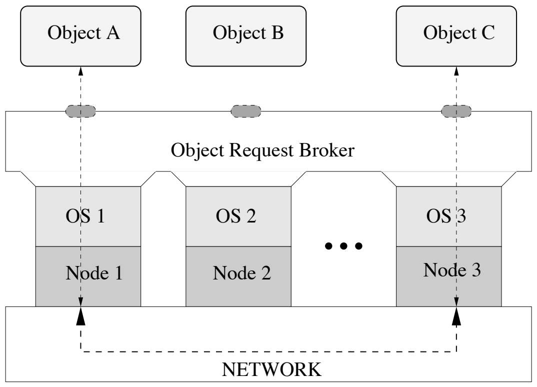

Fig. 2. A middleware : CORBA

The online codes are *objects* which are distributed over a network. But more than a bunch of classes, a complete framework must be designed and created in order to ensure the scalability and extensibility of the design. By framework, we mean a facilitating backbone which combines sets of related components for a specific purpose or domain. A distinguishing feature of a framework is that it defines a generic design that supports a bi-directional flow of control between the application and itself. At first glance extracting the genericity doesn't seem that easy : a given problem is tightly bound to its associated parameter set and the numerical type is also a concern. It is crucial to get rid of these dependencies or at least to weaken them.

BlackBox classes and parameter dependency First, let's have a look at a sketch of the BlackBox classes hierarchy, see figure 3 on the following page. The BlackBox classes are templates which are depending on the numerical type used. The base class `sctkBlackBox` defines an abstract interface for the inherited classes. There is a branch which is not seen here, it is the off-line part of the design since we concentrate only on the on-line side. Then `sctkBlackBoxOnline` provides the abstract interface, mostly inherited from `sctkBlackBox`, for all online code subclasses.

The specialization operated by introducing `sctkBlackBoxOnline` is needed to weaken as much as possible the parameter set dependency, the figure 4 illustrates this. Here `ParameterTypeRef` is a reference to a class representing the parameter set. The idea is to delegate the computation of the $\sigma^q(\mu)$ within the subclasses of `sctkBlackBoxOutputType`. At the instantiation of any subclasses `sctkBlackBoxOnline`, the user has to provide a subclass which will give the $\sigma^q(\mu)$ to the online code for the desired parameter set μ. As one can see the abstract classes `sctkBlackBox`, `sctkBlackBoxOnline` and `sctkBlackBoxOutputType` define a general framework where it is possible to plug a very wide range of *blackbox* methods. So far four methods have been implemented using this framework :

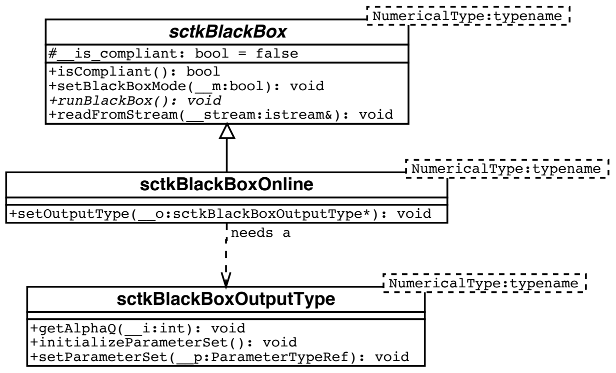
NumericalType:typename
sctkBlackBox
#__is_compliant: bool = false
+isCompliant(): bool
+setBlackBoxMode(__m:bool): void
+runBlackBox(): void
+readFromStream(__stream:istream&): void
NumericalType:typename
sctkBlackBoxOnline
+setOutputType(__o:sctkBlackBoxOutputType*): void
needs a
NumericalType:typename
sctkBlackBoxOutputType
+getAlphaQ(__i:int): void
+initializeParameterSet(): void
+setParameterSet(__p:ParameterTypeRef): void

Fig. 4. UML collaboration diagram: parameter set dependency weakening.

1. `sctkBlackBoxOnlineSimple` is used for simple output computations that don't require the solution of a partial differential equation ;
2. `sctkBlackBoxOnlineStandard` and `sctkBlackBoxOnlineStandardEigen` are two incarnations of the standard blackbox Reduced-Basis output bounds method ;
3. `sctkBlackBoxOnlineOQ` uses a blackbox Reduced-Basis output bounds method of complexity $\mathcal{O}(Q)$ while the previous two had a complexity of $\mathcal{O}(Q^2)$.

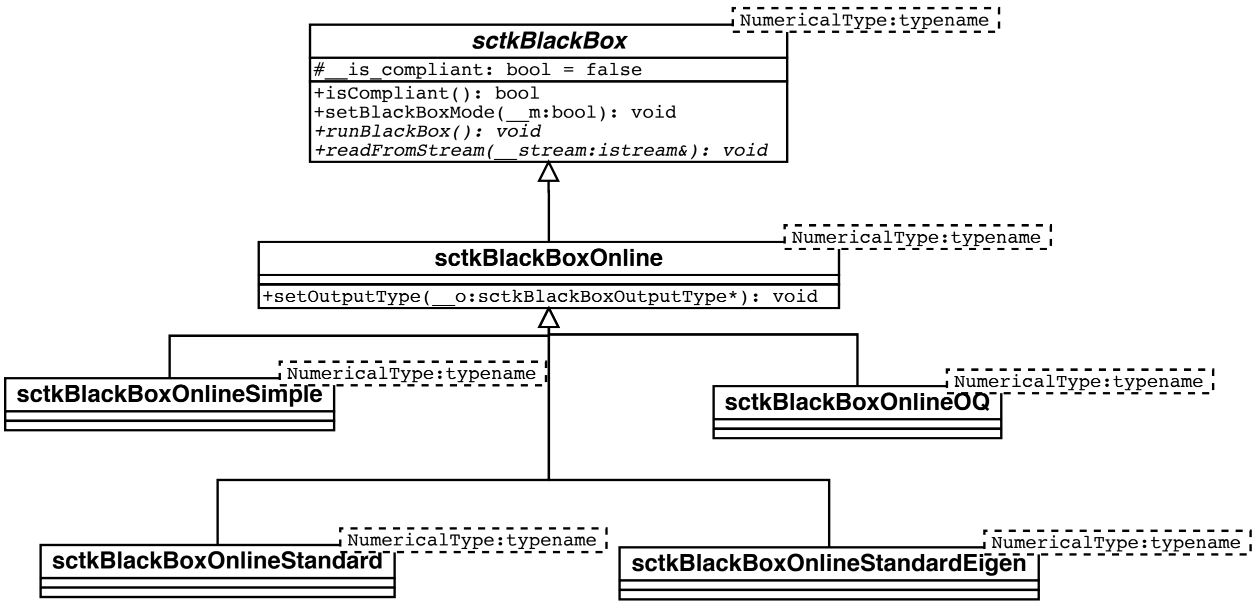

Fig. 3. A UML diagram of BlackBox classes hierarchy.

Note that the design is generic enough to solve a wide range of problems using these four methods provided that the online codes developer writes the necessary subclass of `sctkBlackBoxOutputType` that defines the $\sigma^q(\mu)$ for his problem.

Now that we have briefly shown the design of the online codes, it is time to *attach* them to a CORBA object – `coBlackBox` – whose interface is described using the Interface Definition Language (IDL) and shown in the next section. Once done, we register the newly created CORBA object in the Implementation Repository (IMR) which will keep track of the location of all the available online codes and will permit a transparent access between a client and an online code object.

A simple client code for SimRes We present a small code client for SimRes. First here is the IDL file providing the interface for the CORBA objects. The object interface shown is the one which makes an online code a CORBA object. Note that it is the same for *all* online codes. It is also important to understand that the current design works well from the performance point of view because there is little communication between the client code and a given CORBA object : while a CORBA object and a C++ object are similar in terms of member functions calls, sending a remote message over a network is orders of magnitude larger than the cost of a C++ method invocation. So the following IDL interface fits really well in a CORBA based environment since there are only the parameter set sent from the client to the object and the output and the associated bound gap sent back from the object to the client.

```
typedef sequence<any>  coParameterList;

interface coBlackBox
{
  void setNewParameterSet( in coParameterList  __pset );
  coParameterList getOutput();
  coParameterList getBoundGap();
  void compute();
};
```

Note that the `getOutput()` and `getBoundGap()` member functions return an array of outputs and bound gaps. This is done to ensure good performances when lots of evaluations are required, see the section on SimTEX for an illustration.

Now we can write a small client program. It is not a working example but rather a excerpt of a client which computes the temperature at the root of the 3D fin using SimRes. The extra work to get this example working is minimal and needs a few extra lines of code. The line 1 includes the header file generated by the IDL compiler providing the skeleton of the coBlackBox class on the client side. The line 5-6 look up the object `fin3d_Troot` in the implementation repository and instantiate it on the server side if found. The line 8-9 set a new parameter set and execute the blackbox method. Then the line 11-12 retrieve the output and bound gap which are still sitting on the server side.

```
1   #include <coBlackBox.hpp>
2
3   int
4   main( int argc, char **argv) {
5     CORBA::Object_var __obj = __orb->bind ( "IDL:fin3d_Troot:1.0" );
6     coBlackBox_var __bb = coBlackBox::_narrow( __obj );
7
8     __bb->setNewParameterSet( __pset );
9     __bb->compute();
10
11    coParameterList* __output_pl = __bb->getOutput();
12    coParameterList* __bound_gap_pl = __bb->getBoundGap();
13  }
```

One can see that the amount of code is very small and that the CORBA objects are indeed objects in the C++ sense, that is they are used, in our case `__bb`, like common C++ objects.

Now we present a more complete client for SimRes.

An interface for SimRes : SimTEX

SimTEX provides an easy while powerful interface to the online code repository. As mentioned earlier, we want also an interface which is portable and which provides a somewhat standard way of presenting scientific results. To achieve this, we chose to use LaTeX with PDF as its output. LaTeX is a standard in scientific typesetting while PDF provides a minimal set of features to create a user interface. If you want to get a first feeling of the interface have a look at the equation 6. Since there are little data manipulated – the parameter set, the output and its associated bound gap – the interface can be quite simple and even reduced to the form of an *actionable equation*. We wrote a special environment for LaTeX called `acteq` for this purpose. It is mainly based on the `hyperref` package. It uses the CGI-like[3] capabilities of PDF, that is there is a button in the interface which, once clicked, will send the form to a web server CGI script that will parse it, instantiate and execute the CORBA online codes using SimRes, and eventually generate PDF graphics if interval values are given for one or two parameters. The figure 5 on the following page shows the UML sequence diagram which describes the data flow when the user click on the = sign.

The `acteq` environment The example's code of the actionable equation shown in the section 3 on the next page is the following :

```
\begin{acteq}{augustine.mit.edu}{Troot:Ttip:Volume}{fin3d}{7}{3}{10}
  \begin{equation*}\left(\begin{array}{r}
        \defoutputchecked{Troot}{T_\mathrm{root}}\vspace{3mm}\\
        \defoutput{Ttip}{T_\mathrm{tip}}    \vspace{3mm}\\
        \defoutput{Volume}{\mbox{Volume}}
      \end{array}   \right)  = \mathcal{F} \left(  \begin{array}{r}
        \param{k^1=}{0.4} \\
        \param{k^2=}{0.6} \\
        \param{k^3=}{0.8} \\
```

[3] CGI stands for Common Gateway Interface

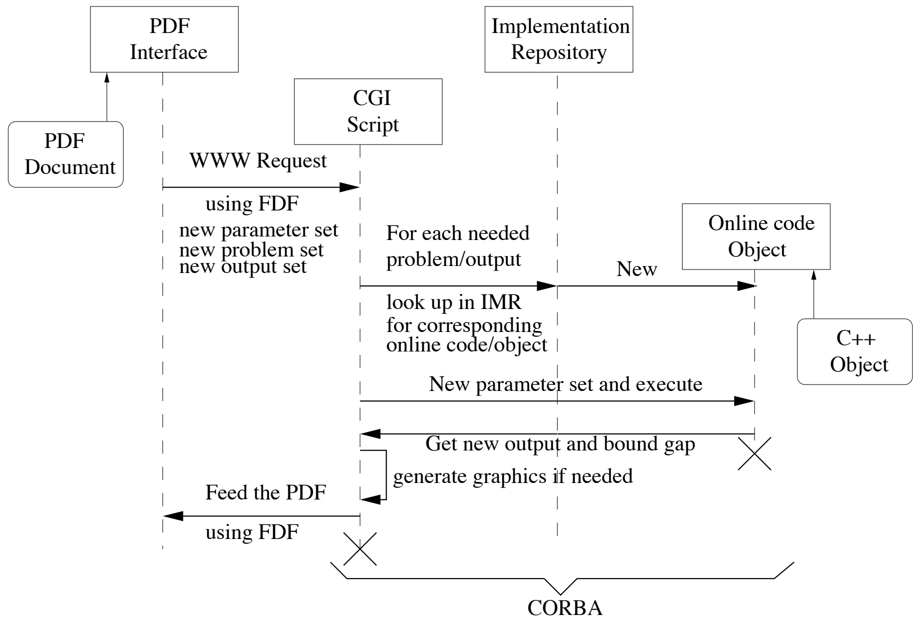
PDF Interface
Implementation Repository
CGI Script
PDF Document
WWW Request
using FDF
new parameter set
new problem set
new output set
For each needed problem/output
New
Online code Object
look up in IMR for corresponding online code/object
C++ Object
New parameter set and execute
Get new output and bound gap
generate graphics if needed
Feed the PDF
using FDF
CORBA

Fig. 5. Sequence disgram of SimTeX

```
        \param{k^4=}{1.2} \\
        \param{\mbox{Bi}=}{0.1} \\
        \param{t=}{0.3} \\
        \param{L=}{2.8}
      \end{array} \right) \,
    \equalsign \, \predoutput \pm \prederror
  \end{equation*} \end{acteq}
```

As you can see the macro system is very simple and requires a minimal knowledge from the writer point of view. The information required is the following :

1. the implementation repository (IMR) of online code, in our case it is `augustine.mit.edu`. This server contains the information about the location of the different available online codes and how to execute them.
2. the available outputs for a given problem. In our case only three outputs are available, the mean temperature at the root of the fin, `Troot`, the mean temperature at the top of the fin, `Ttip`, and the volume of the fin, `Volume`.
3. the name of the problem, in our case `fin3d`.
4. the number of parameters for this problem, in our case 7.
5. the number of outputs, in our case 3.
6. the last parameter of the `acteq` environment is just the number of points which are generated if a interval value is given for a parameter.

This system, while simple on the user side, has proven to be quite helpful and powerful. It has been used successfully as a research tool, as an educational tool and also as a presentation tool. Now let's see the whole system is action.

3 An example : the 3D fin

In this example we consider a three-dimensional thermal fin designed to effectively remove heat from a surface. The three-dimensional fin, shown in Figure 6 on the facing page, consists of a vertical central "post" and four horizontal "subfins"; the fin conducts heat from a prescribed uniform flux "source" at the root, $\Gamma_{\rm root}$, through the large-surface-area subfins to surrounding flowing air.

The fin is characterized by a seven–component parameter vector, or "input," $\mu = (\mu^1, \mu^2, \ldots, \mu^7)$, where $\mu^i = k^i$, $i = 1, \ldots, 4$, $\mu^5 = \mathrm{Bi}$, $\mu^6 = L$, and $\mu^7 = t$; μ may take on any value in a specified design space $\mathcal{D} \subset \mathbb{R}^7$. Here k^i is the thermal conductivity of the i^{th} subfin (normalized relative to the post conductivity $k^0 \equiv 1$); Bi is the Biot number, a nondimensional heat transfer coefficient reflecting convective transport to the air at the fin surfaces; and L and t are the length and thickness of the subfins (normalized relative to the post width). The fin is one unit deep(the root is square) and four units tall.

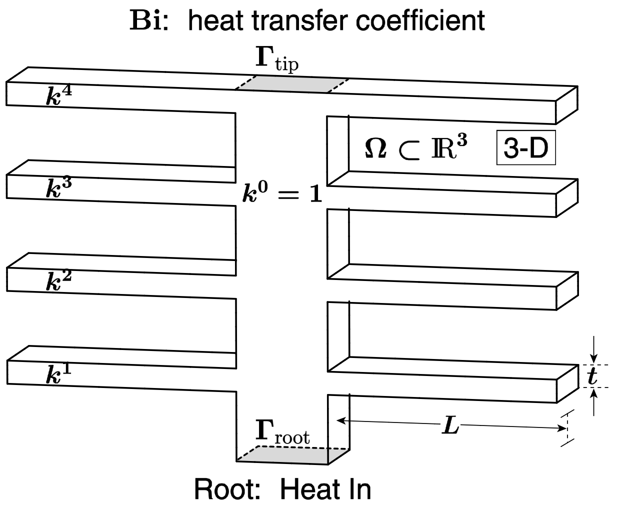

Fig. 6. 3D Thermal Fin

We consider several outputs of interest. The first output is the performance metric, $T_{\rm root} \in \mathbb{R}$, is taken to be the average temperature of the fin root normalized by the prescribed heat flux into the fin root. This output relates directly to the cooling efficiency of the fin — lower values of $T_{\rm root}$ imply better performance. The second output is the average temperature at the tip of the fin, $T_{\rm tip} \in \mathbb{R}$. Low values of $T_{\rm tip}$ indicate that the cooling is effected primarily by the lowest subfins — hence the upper subfins are wasted material. The last output is the volume of the fin which represents weight and material cost — thus lower values are preferred. In order to optimize the design, we must be able to rapidly evaluate $T_{\rm root}(\mu)$, $T_{\rm tip}(\mu)$ and the volume of the fin for a large number of parameter values $\mu \in \mathcal{D}$.

The steady–state temperature distribution within the fin, $u(\mathbf{x})$, is governed by the elliptic partial differential equation

$$-k^i\,\nabla^2 u^i = 0 \text{ in } \Omega^i,\ i = 0, \ldots, 4,$$

where ∇^2 is the Laplacian operator, and u^i refers to the restriction of u to Ω^i. Here Ω^i is the region of the fin with conductivity k^i, $i = 0, \ldots, 4$: Ω^0 is thus the central post, and Ω^i, $i = 1, \ldots, 4$, corresponds to the four subfins. We must also ensure continuity of temperature and heat flux at the conductivity–discontinuity interfaces $\Gamma^i \equiv \partial\Omega^0 \cap \partial\Omega^i$, $i = 1, \ldots, 4$, where $\partial\Omega^i$ denotes the boundary of Ω^i:

$$\left.\begin{array}{c} u^0 = u^i \\ -(\nabla u^0 \cdot \hat{\mathbf{n}}^i) = -k^i(\nabla u^i \cdot \hat{\mathbf{n}}^i) \end{array}\right\} \text{ on } \Gamma^i,\ i = 1, \ldots, 4;$$

here $\hat{\mathbf{n}}^i$ is the outward normal on $\partial\Omega^i$. Finally, we introduce a Neumann flux boundary condition on the fin root

$$-(\nabla u^0 \cdot \hat{\mathbf{n}}^0) = -1 \text{ on } \Gamma_{\rm root},$$

which models the heat source; and a Robin boundary condition

$$-k^i(\nabla u^i \cdot \hat{\mathbf{n}}^i) = \mathrm{Bi}\ u^i \text{ on } \Gamma^i_{\rm ext},\ i = 0, \ldots, 4,$$

which models the convective heat losses. Here $\Gamma^i_{\rm ext}$ is that part of the boundary of Ω^i exposed to the fluid that is $\partial\Omega \setminus \Gamma_{\rm root}$.

For every choice of the design parameter-vector μ — which determines the k^i, Bi, and also the fin geometry through L and t — solution of the above system of equations yields the temperature distribution $u(\mathbf{x}; \mu)$ in the fin. The outputs of interest — the average temperature at the root, the average temperature at the tip — can be expressed respectively as $T_{\rm root}(\mu) = \ell^O_{\rm root}(u(\mathbf{x}; \mu))$, $T_{\rm tip}(\mu) = \ell^O_{\rm tip}(u(\mathbf{x}; \mu))$, where

$$\ell^O_{\rm root}(v) = \int_{\Gamma_{\rm root}} v, \quad \ell^O_{\rm tip}(v) = \int_{\Gamma_{\rm tip}} v,$$

(recall Γ_{root} and Γ_{tip} are of area unity). As for the volume, it is given by the following formula

$$\text{Volume} = 4 + 8\,L\,t.$$

These functional dependencies can, in turn, be summarized by the input-output relationship

$$\begin{pmatrix} T_{\text{root}} & \boxed{\checkmark} \\ T_{\text{tip}} & \boxed{\checkmark} \\ \text{Volume} & \boxed{} \end{pmatrix} = \mathcal{F} \begin{pmatrix} k^1 = & \boxed{0.4} \\ k^2 = & \boxed{0.6} \\ k^3 = & \boxed{0.8} \\ k^4 = & \boxed{1.2} \\ \text{Bi} = & \boxed{0.1} \\ t = & \boxed{0.3} \\ L = & \boxed{2.8} \end{pmatrix} \boxed{=} \boxed{} \pm \boxed{} \tag{6}$$

FIGURE

For our parameter space we choose $\mathcal{D} = [0.1, 10.0]^4 \times [0.01, 1.0] \times [0.1, 0.5] \times [2.0 \times 3.0]$, that is, $0.1 \leq k^i \leq 10.0$, $i = 1, \ldots, 4$ for the conductivities, $0.01 \leq \text{Bi} \leq 1.0$ for the Biot number, and $0.1 \leq t \leq 0.5$, $2.0 \leq L \leq 3.0$ for the geometric parameters. [4] In general, better performance — lower temperature — requires larger fin volume (e.g., larger t) or materials with higher conductivity; in both cases the production cost of the fin would increase accordingly. Hence there are design trade-offs that must be investigated.

4 Conclusion

This framework is likely to be extended in the next few months, and will certainly enjoy many improvements. It has already been tested on a few problems already available online on our web site http://augustine.mit.edu through the SimTeX interface. The component-wise design of the overall system is very flexible, scales well as the number of online codes increases and is an elegant solution as a platform for engineering design, research and education.

Acknowledgements This work was supported by the Singapore-MIT Alliance, by AFOSR Grant F49620-97-1-0052, and by NASA Grant NAG1-1978.

References

1. B. O. Almroth, P. Stern, and F. A. Brogan. Automatic choice of global shape functions in structural analysis. *AIAA Journal*, 16:525–528, May 1978.
2. Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley professional computing series. Addison Wesley, 1999.
3. L. Machiels, Y. Maday, I. B. Oliveira, A. T. Patera, and D. V. Rovas. Output bounds for reduced-basis approximations of symmetric positive definite eigenvalue problems. *C. R. Acad. Sci. Paris, Série I*, 2000.
4. Y. Maday, A. T. Patera, and D. V. Rovas. A blackbox reduced-basis output bound method for noncoercive linear problems. *College de France Series; also MIT-FML Report 00-2-1*, 2000.
5. D. A. Nagy. Modal representation of geometrically nonlinear behaviour by the finite element method. *Computers and Structures*, 10:683–688, 1979.
6. A. K. Noor and J. M. Peters. Reduced basis technique for nonlinear analysis of structures. *AIAA Journal*, 18(4):455–462, April 1980.

[4] At present the system does not verify ranges; for meaningful results, make sure that each parameter component is within the values indicated. Values outside the specified ranges (but still honoring $k^i > 0$, $i = 1, \ldots, 4$, $\text{Bi} > 0$, $t > 0$, $L > 0$) will continue to give correct results, but in most cases the error bars will be unacceptably large.