Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

INTRODUCING VIRTUAL ENGINEERING TECHNOLOGY INTO INTERACTIVE DESIGN PROCESS WITH HIGH-FIDELITY MODELS

Gengxun Huang

Mechanical Engineering Department
2274 Howe Hall
Iowa State University
Ames, IA 50010, U.S.A.

Kenneth Mark Bryden

Mechanical Engineering Department
3030 Black Engineering
Iowa State University
Ames, IA 50010, U.S.A.

ABSTRACT

Product design is a complex decision-making process requiring intense interaction between designers and the designed product. Consequently, the design process is significantly different from a pure mathematical optimization. This paper presents a decision support platform for interactive design that integrates mathematical optimization with human interaction based on VE-Suite. Current efforts are geared toward seamlessly linking high fidelity models, numerical optimization and human interaction to improve efficiency and quality in system performance. The designer's interaction causes the optimization process to dynamically change by adding, deleting, and modifying objectives, constraints, and other parameters that govern the process. As an illustration, a coal pipe design case is used to demonstrate the new platform's capabilities. The case has demonstrated that adding user interaction into the design process has the potential to improve design efficiency and quality.

1 INTRODUCTION

Today, there are numerous research and software packages available in the literature. In general, machine-based numerical optimization has significant utility within well-defined routines and detailed design domains. In these systems the stereotypical role of the designer is to specify the problem including predefined constraints and control parameters, and then initiate a computer search to find an optimal solution. One of the major drawbacks of this type of design system is that this approach neglects the important aspect of the optimization task which is to obtain useable solutions. In practical applications, it is the solution suggested by the optimizer, not the actual details of the design, that are most interesting. Anderson (2000) pointed out that in many contexts interaction is more important than efficiency since the optimization algorithm is working with an impoverished objective function, and the ability to successfully implement the solution depends on how well people understand and trust it. Users must understand and trust the generated solutions to use them effectively. Those limitations have led Vladimir (2002) to report that numerical optimization technology still has very limited success in an industrial design environment.

In recent years research related to user-based interactive optimization has increased rapidly. More and more researchers agree that users are more likely to understand a solution that they helped to create than one that is simply presented to them. Research shows that including humans "in-the-loop" during the design process can enhance optimization performance. However, most literature in this field focuses on extreme cases. For example, some interactive optimization algorithms require user guidance at every step. Without user instruction, such algorithms cannot function. Therefore, they are typically only considered in cases where it is hard or impossible to use numerical models to represent the problems such as in works of art (Hong 2004). Understandably, given the demands on user time imposed by algorithms of this kind, these algorithms are seldom used in the engineering design process.

Other researchers used computational steering technology to interactively control a computational process, such as CFD simulation and optimization, during execution (i.e. SCIRun 2005). With computational steering, users are continuously provided with visual feedback about the state of their simulation and can change parameters on the fly. This allows designers to modify parameters in order to optimize their product. Although the concept of computational steering is powerful, the implementation of computational steering is very difficult. It requires knowledge of simulation, visualization, user interfacing, and data communication (Xiao and Bryden 2004). Therefore, while computational steering technologies offer significant capabilities, they are generally time consuming to use and hard to adapt to meet the actual project requirements. For example, to integrate an existing application into these systems, the application source code has to be manually annotated with program statements by the application developer.

From these experiences, we observe that there are two key issues in designing interactive engineering systems. First, an appropriate division of labour between humans and computers is needed so that humans' superior abstract thinking and computers' superior computational speed can work together to produce a better performance than either could do alone. The extent to which human interaction should influence the optimization process is still unclear. Fortunately, end-users usually have more experience handling this issue if the system allows the user to guide the design process. Secondly, in order for the system to gain acceptance, it should be easy to use and provide real-world usability. Based on these two key requirements, this system should have the following characteristics:

1. Scalability — enable users from different areas to easily build applications inside the system or add new capabilities without dealing with system programming issues.
2. Extensibility— enable the system to grow by extending existing capabilities and adding new technologies.
3. Flexibility — enable users to choose from a variety of solvers and other computer aided engineering tools in a platform independent manner.
4. Physically-based, real-time visualization— enable users to observer the analysis result in a realistic and intuitive manner.

Currently, there is no system available that meets these requirements. However, virtual engineering technology, which is defined as a technology that integrates and combines geometric models, analysis, simulation, optimization and other decision making tools within a virtual environment to facilitate multidisciplinary and collaborative product realization (Xiao and Bryden 2004), can be used as a way of gaining insight into the design space. Furthermore, virtual engineering technology can also be used to quantitatively and qualitatively identify innovative design options, which is exactly what interactive design system requires. Hence, we have worked to extend the virtual design and engineering capabilities of VE_Suite to interactive product design. VE_Suite is an open source virtual engineering software package that is currently under active development by the complex system virtual engineering group at Iowa State University. VE_Suite serves as a high-level support tool for engineers who want to transform their traditional applications into virtual engineering applications.

The architecture of VE-Suite is shown in Figure 1. The core modules of VE-Suite are VE_Xplorer (the graphical engine), VE_Conductor (the GUI front end to the virtual engineering framework), and VE_CE (the computational engine). VE-Suite divides the implementation of virtual engineering into two tasks: WxWidgets based user interfaces and the computational unit (see Figure 1). These two tasks comprise the VE_Suite API. VE_Suite is general in nature and the three key components can run on a geographically diverse set of heterogeneous computer platforms. For example, the VE_CE component can run on a Linux cluster; the VE_Xplorer component can run on an SGI machine; and VE_Conductor can run on a portable Tablet PC. Also, the three core components of VE-Suite can function as complete stand-alone applications provided that the necessary input files are prepared by the user. This feature ensures that VE-Suite can be used in a variety of applications. Additional details about VE_Suite can be found in McCorkle and Bryden's report (2003).

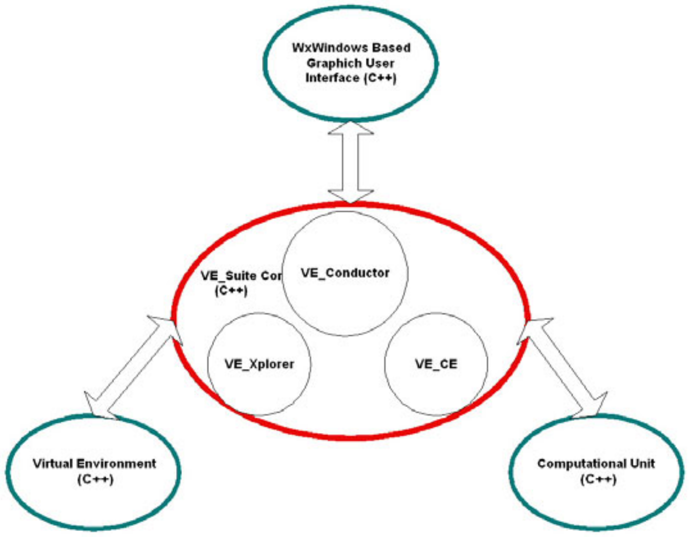

WxWindows Based Graphich User Interface (C++)
VE_Suite Cor (C++)
VE_Conductor
VE_Xplorer
VE_CE
Virtual Environment (C++)
Computational Unit (C++)

Figure 1: VE_Suite Architecture

This paper illustrates this new interactive decision support system using evolutionary algorithms (EAs) as the optimization algorithm, computational fluid dynamic (CFD) as the evaluation mechanism, and VE_Suite as the backbone of the system. In Section 2 we introduce the necessary background of the API provided by VE_Suite. Section 3 contains a more detailed presentation of building a new interactive design system using the VE_Suite API. Emphasis is placed on building the computational unit and the plug-in GUI. In Section 4, the coal pipe design experiments are presented to demonstrate the platform's performance. In the final section, we discuss what has been accomplished to date and present directions for future work.

2 VE_SUITE CORE CHARACTERISTIC AND ITS API

To enable performance on multiple operating systems and immersive technology platforms, VE-Suite is built upon

VR Juggler <http://www.vrjuggler.org>, SGI OpenGL Performer <http://www.sgi.com/products/software/performer>, and Kitware's VTK <http://public.kitware.com/VTK>. The communication between the different components and user-defined modules is built upon the widely adapted and stable Common Object Request Broker Architecture (CORBA) <http://www.omg.org> standard developed over the last decade. The Executive module (one of the key modules in the VE_Suite Core) implements two of the standard CORBA services bundled with the TAO CORBA<http://www.ece.uci.edu./~schmidt/TAO.html> distribution. The first service is the COSS Naming Service that is used as a lookup table of the currently running processes which allows clients to find running process based on the given ID. The second service is the interface that houses the functional and data type definitions and provides them graphically to the user to allow him/her to define a workflow in the graphical interface. In our system, the running process (usually the computational unit) can broadcast its status and analysis information to multiple GUI clients. Any given GUI can be connected into the system information stream at any point in time to view the current state of the running process. The system was designed to allow the GUI to be shutdown and restarted at will without any impact on the computational unit's execution. This attach/detach functionality gives the user the ability to monitor the design process easily. As an example, this functionality would allow a user to build and start a simulation and then detach from the computational engine. The user could then go to a different location, reattach to the running process, and regain monitoring and control functions. The other advantage of the use of component architecture design techniques is that multiple GUIs can also be connected simultaneously from different computers and allow multiple users to monitor a simulation from different locations.

The VE-Suite core provides a simple API for the user to build his own interface to fit the requirement of the actual application. wxWidgets <http://www.wxWindows.org> is provided as the GUI library for the user to use since it is easy to learn (C++ based), and well-maintained. A plug-in C++ base class (Figure 2) defining the basic GUI interface is provided to all module developers. Developers can create their own GUIs and then compile the resulting code into a DLL in Windows or shared libraries in Linux/Unix. VE-Suite is able to dynamically discover, identify, and load the user's GUI from these shared libraries. Since this user-defined GUI is inherited from the plug-in base class, it has all functionality provided by the framework. This detachable GUI is where the user is able to create the design configuration, set model inputs, start and stop execution of simulation, and view simulation results. Also, it is more likely to meet user specific requirements because the interface is built by the person who will use it. This useable interface will be custom-tailored to the user's demands and can easily be adapted if the user's focus of interests changes. In order to transfer data from the GUI to its control, the function TransferDataFromWindow in the base class should be overriden. Figure 3 illustrates a simple, yet complete example of how to transfer data from GUI components (wxTextCtrl) to its own control.

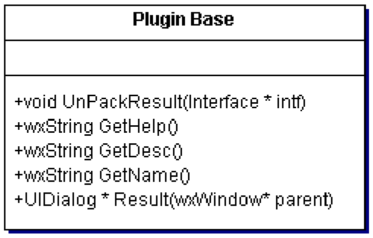
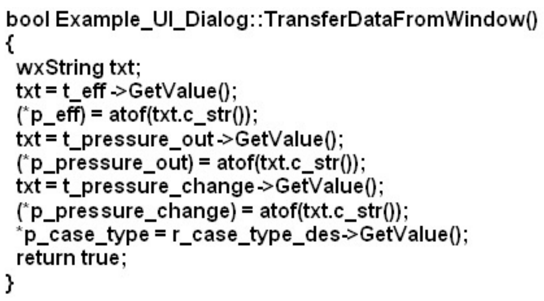

Plugin Base
+void UnPackResult(Interface * intf)
+wxString GetHelp()
+wxString GetDesc()
+wxString GetName()
+UIDialog * Result(wxWindow* parent)

Figure 2: PluginBase Class



```
bool Example_UI_Dialog::TransferDataFromWindow()
{
 wxString txt;
 txt = t_eff->GetValue();
 (*p_eff) = atof(txt.c_str());
 txt = t_pressure_out->GetValue();
 (*p_pressure_out) = atof(txt.c_str());
 txt = t_pressure_change->GetValue();
 (*p_pressure_change) = atof(txt.c_str());
 *p_case_type = r_case_type_des->GetValue();
 return true;
}
```

Figure 3: Example of TransferDataFromWindow

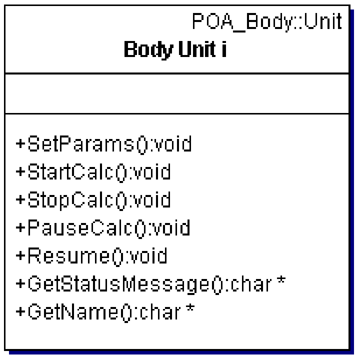
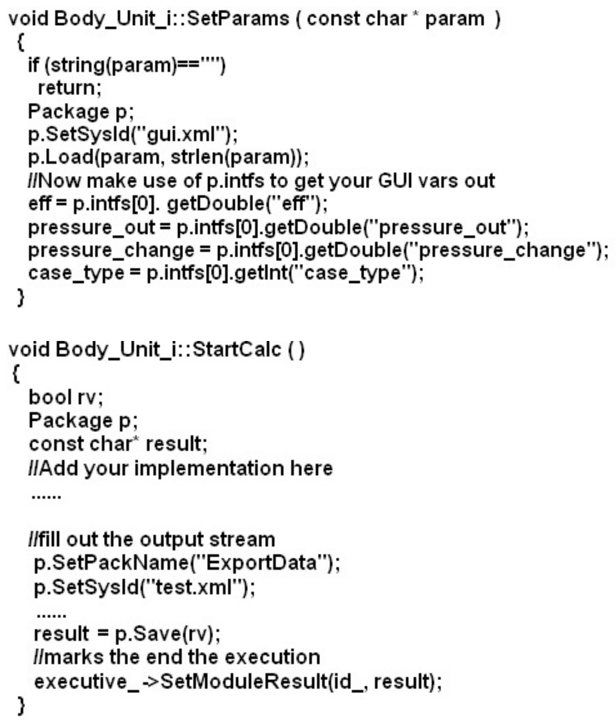


Figure 4: Body_Unit_i Class

Since users from different fields have a wide variety of needs, the analysis tools and their use require a detailed understanding of the problem. VE-Suite uses the Computational engine (VE_CE) module to construct, coordinate, schedule, and monitor the running processes. Because of this, existing commercial, in-house, and open source analysis packages can be used directly with VE_Suite. The analysis package can vary from Microsoft Excel^{TM} spread sheets to process models to CFD models. The analyst only needs to provide a few routines to declare the communication variables. VE_CE provides a CORBA server with which the detachable GUI connects. It is capable of running a simulation containing a multitude of different types of models, each accepting and generating a myriad of data types. Once a client-server connection is made, the unit is able to send results, messages, updates, and communications from other attached GUIs in real time.



```
void Body_Unit_i::SetParams ( const char * param )
 {
  if (string(param)=="")
   return;
  Package p;
  p.SetSysId("gui.xml");
  p.Load(param, strlen(param));
  //Now make use of p.intfs to get your GUI vars out
  eff = p.intfs[0]. getDouble("eff");
  pressure_out = p.intfs[0].getDouble("pressure_out");
  pressure_change = p.intfs[0].getDouble("pressure_change");
  case_type = p.intfs[0].getInt("case_type");
 }

void Body_Unit_i::StartCalc ( )
{
  bool rv;
  Package p;
  const char* result;
  //Add your implementation here
  ......

  //fill out the output stream
   p.SetPackName("ExportData");
   p.SetSysId("test.xml");
   ......
   result = p.Save(rv);
   //marks the end the execution
   executive_->SetModuleResult(id_, result);
 }
```

Figure 5: Template for SetParams and StartCalc

VE-Suite provides a template for users to fill their own computation units into the VE-Suite framework. Figure 4 shows the basic functions in Body_Unit_i class. Figure 5 shows two important functions that users need to modify in order to implement their own applications. First, SetParams needs to be implemented for the user's customized module. This call takes the GUI inputs passed by the CORBA interface (Executive Module). Using Figure 3 as example GUI inputs, every GUI variable that is needed can be retrieved from the interface by calling p.intfs[0].get***("paramname") (Note: *** represents the data type such as Double, Int). Additional details of how to add existing applications are discussed in Section 3 using the new design platform as an example.

A key aim of virtual engineering is to fully engage the human capacity for problem solving by creating a realistic experience for the user so that s/he can focus entirely on the engineering problem. The advantage is that previously indescribable complexities can be understood and the full range of engineering solutions can be explored. The graphical engine (VE-Xplorer) provides the core functionality for the virtual engineering aspect of the framework. It can load geometry files, three-dimensional simulation data and experimental data of almost every format into a scene. VR Juggler is used to handle interfacing with VR hardware and graphics rendering platform. VE-Suite handles the creation of the virtual environment and VR Juggler allows software to run with any type of virtual environment, from a regular 2-D screen to a six-walled immersive virtual space. Due to the generality of the visualization requirements, the VE-Suite core provides the basic visualization GUI so that users can navigate and control the scene. The GUI is laid out in a tabbed notebook format as shown in Figure 6. Typical operations can be performed on each tab of the GUI. For example, the navigation tab is the main window for users to navigate through the scene and choose the location he or she wishes to observe the data from. Through the visualization tab, users can select different visualization methods (contour surface, vector fields, etc.) to visualize the fluid field. Figure 7 shows the streamline of flow through a 90 degree elbow. VE_Xplorer is designed such that users need not know the details of graphics and virtual reality programming. It should be noted that VE-Suite also provides an interface that allows advanced users to add or modify the existing functions or visualization GUI.

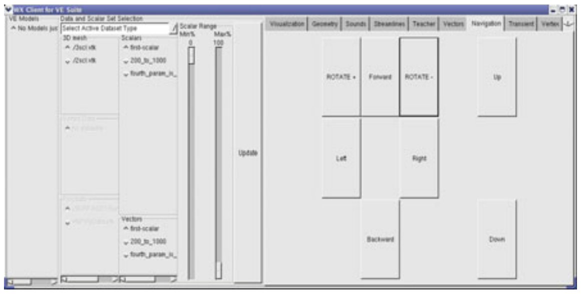


Figure 6: VE_Xplorer GUI

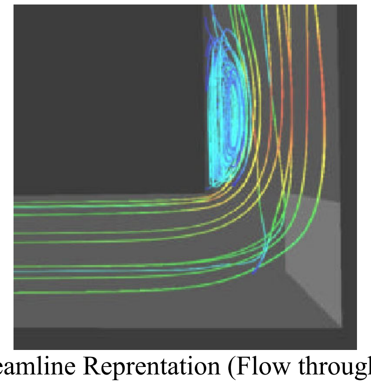


Figure 7: Streamline Reprentation (Flow through Elbow)

3 PROPOSED INTERACTIVE EVOLUTIONARY DESIGN PLATFORM

As discussed in Section 1, to allow effective user interaction, the design system should provide users with data and other information about the designed product and its performance in an understandable and intuitive way. In addition, the design system should also give users the ability to exert influence on the process in an intuitive manner. VE-Suite already provides this basic functionality, such as detachable GUI, realistic graphical engine and bi-directional bindings between the GUI and the computational unit, which are necessary to meet the requirement of an interactive evolutionary design system. Therefore, the implementation of the proposed interactive design platform is very straightforward. Figure 8 shows the basic information flow of this system. As shown, the primary component in the system that users will interact with is the plug-in GUI. Using VE-Suite, designers can not only access the traditional design information, such as tables, two-dimensional plots and colour codes, whenever they want, but also three-dimensional virtual images (i.e. streamline) from high fidelity datasets such as CFD data. On the computational unit side, the main module connects with the CORBA interface to accept the user's input. A multi-threaded approach is used in order to minimize the effort of adapting the existing computational source code into an interactive design space. A centralized Database Manager module is used for interpreting the user's request, data management and transfer. This additional Database Manager module provides a simple way to track the state of the running process and exchange information between the Model thread and View thread. The following sections will focus on the implementation of the computational unit and the plug-in GUI design.

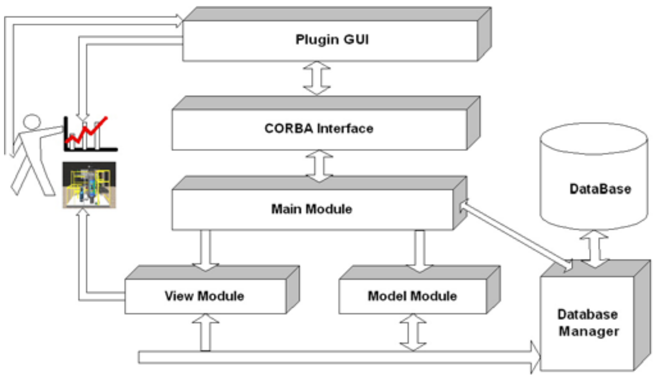


Figure 8: Diagram of the System Architecture

3.1 Computational Unit (Optimizer)

In the area of engineering analysis tools, there has been a growing interest in optimal design in conjunction with CFD in the field of fluid mechanics. In many cases, improving the thermal fluid system design requires understanding precise details of the fluid flow, the heat transfer, or other phenomena of interest. High-fidelity models such as CFD can provide a greater level of detail and improved solutions compared to simple engineering relationships. Therefore, thermal system design can be improved if CFD analysis is used instead of simplified modelling techniques such as the Moody diagram, heat loss coefficients, etc. However, it is not uncommon for one CFD analysis case to take several days to complete. Therefore, traditionally, it is extremely challenging to add human interaction into the system which uses high-fidelity models such as CFD solvers as an evaluation mechanism because of the difference in time between human thinking and machine problem solving. On the other hand, this system happens to be the system that needs human interaction most, especially when this system is used for conceptual or preliminary design. For instance, if a constraint is omitted or an assumption is incorrectly made, numerical optimizers will often return a physically unreasonable design or will fail to converge after several days. However, when the user is allowed to monitor the design evolution, it is more likely that the user will catch the design flaw much earlier. Currently, there is no established standard to determine how much user time and expertise need to be devoted to steering the search process in order for the added user interaction to be most profitable. However, in the case of using high fidelity solver as the analysis code, compared to the CPU time needed for the CFD solver, the time used for the operation of human interaction is minimal. Therefore, adding human interaction into the optimization process can be generally expected to be helpful.

The optimization algorithm (computational unit) we propose here lies somewhere between two extremes as discussed in Section 1 in the sense of adding human interaction into the design process. Using our framework, the user does not determine the behaviour of the computational process. Therefore it is still a machine-based optimization, but users are given the capability to influence the process by altering system parameters or changing objective functions.

The computational unit for this interactive design system utilizes a standard Evolutionary Algorithms optimization software. The standard EA optimizer includes several components: selection, crossover, mutation, and replacement. For more information about EAs see Goldberg (1989). This core function (the Evolve function in the Model class in Figure 9) becomes the central process of the computational unit. The template class Body_Unit_i provides necessary CORBA interfaces required for framework integration and communication. Therefore, a simple App class is added as an interface between the Model class and the Body_Unit_i class. A classical Subject/Observer design pattern is used among these three classes (See Figure 9). This is mainly used when new results need to be sent back to the plug-in GUI so that users can assess the latest design information. In our approach, the App class acts as a mediator between the other two classes; it is the observer of Model and subject of Body_Unit_i. As a subject of App, Model will notify its observer App that new result is ready for user to pick up. As a subject of Body_Unit_i, in App's Update function, it notifies Body_Unit_i and passes the information to Body_Unit_i. This Subject/Observer/ Mediator trio is commonly known as the Model View Controller (MVC) pattern. The advantage of this method is that the Model class can be almost the same as before. As mentioned previously, a multi-threaded approach is used in which the Model thread executes the optimization loop and runs the analysis routines through the Evolve function in Figure 9, and the View thread dynamically displays the evolutions of the system according to users' requests. The system is designed in this way so that the main optimization loop will not be interrupted when users inspect the complex physical-based images.

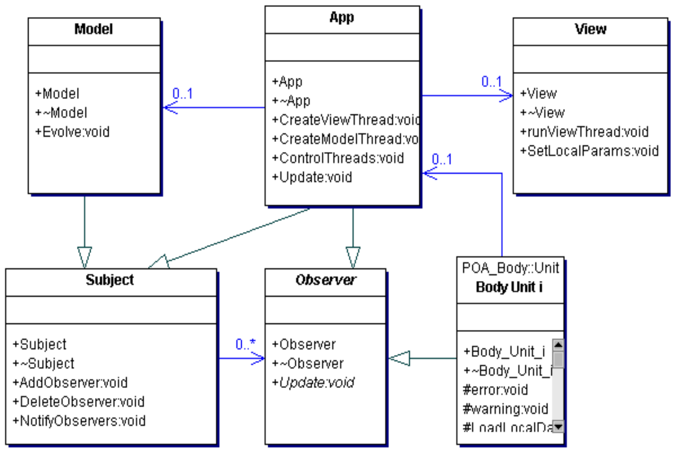
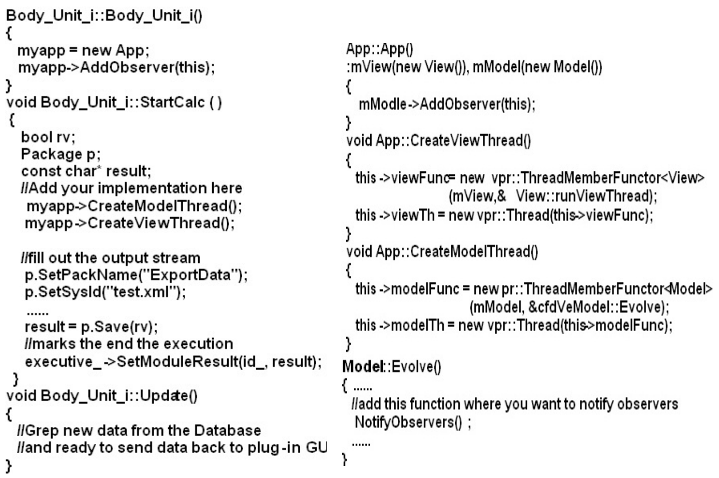


Figure 9: Computational Unit Scheme


```
Body_Unit_i::Body_Unit_i()
{
  myapp = new App;
  myapp->AddObserver(this);
}
void Body_Unit_i::StartCalc ( )
 {
   bool rv;
   Package p;
   const char* result;
   //Add your implementation here
    myapp->CreateModelThread();
    myapp->CreateViewThread();

   //fill out the output stream
    p.SetPackName("ExportData");
    p.SetSysId("test.xml");
    ......
    result = p.Save(rv);
    //marks the end the execution
    executive_->SetModuleResult(id_, result);
 }
void Body_Unit_i::Update()
{
  //Grep new data from the Database
  //and ready to send data back to plug-in GU
}
```

```
App::App()
:mView(new View()), mModel(new Model())
{
  mModle->AddObserver(this);
}
void App::CreateViewThread()
{
 this ->viewFunc= new  vpr::ThreadMemberFunctor<View>
                  (mView,&  View::runViewThread);
 this ->viewTh = new vpr::Thread(this->viewFunc);
}
void App::CreateModelThread()
{
 this ->modelFunc = new pr::ThreadMemberFunctor<Model>
                     (mModel, &cfdVeModel::Evolve);
 this ->modelTh = new vpr::Thread(this->modelFunc);
}
Model::Evolve()
{ ......
 //add this function where you want to notify observers
  NotifyObservers() ;
  ......
}
```

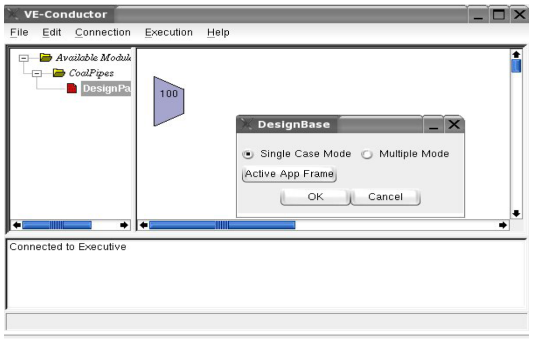
Figure 10: Computational Unit Application

Figure 10 shows the implementation of these three classes. Notice, the user-defined Model class needs to inherit from Subject class and call the NotifyObservers function in the Evolve function; this function notifies all observers that new data is available. In our case, since it may be of interest to the designer to visualize the path taken by the EAs optimizer, the NotifyObservers function is added at the end of each mating event loop. Therefore, users can always have access to the newest information such as the fitness and new creature's geometry whenever they check the result from GUI. Also, threading is done using the high-level threading API provided in VPR, the VR Juggler Portable Runtime Library. It is worth mentioning that there is an external TCP/IP Socket (VPR-based) connection between the View module and VE_Xplorer (VE-Suite graphical engine). This connection allows large high-fidelity datasets to be transferred to the graphical engineering without interrupting the overall network communication.

3.2 Plug-in GUI

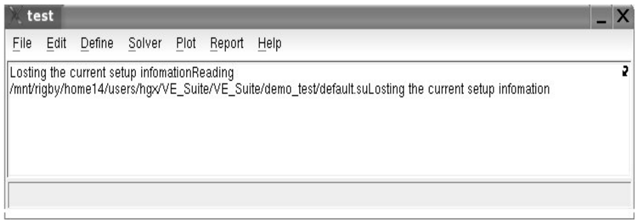
Since the goal of our study is to set up an interactive design system that allows users to provide input parameters to the underlying high fidelity analysis models, observe the analysis result, and optimize the product design, the graphical user interface must be easy to use. Over the course of development of GUI, it was found that certain features are required. For example, the ability to save the current case, reload the existing case file, basic EAs control parameters setup, etc. Although the current GUI is still in its initial stage, it has the basic functionality such as selecting active design parameters, defining EA parameters, defining constraints, and selecting initial design candidates. Figure 11 shows the MainFrame provided by VE_Conductor. It provides the method to connect to the CORBA interface. The main window for the user-defined plug-in GUI is shown in Figure 12. From this main window, the user is able to access the active "system setup"dialog (Figure 13), "Active Design Params Setup" dialog (Figure 14) and "EAs Params Setup" dialog, etc. Users can design the design case using this GUI or load necessary information from an existing file. The current GUI also provides a method for the user to set up the initial populations (See Figure 15). Once the task is completely-filled out in the VE_Condutor's Executation menu (Figure 11), the task can be submitted to the computational unit defined by the user across a network. Usually, an optimization not only runs results about the solutions to a specific problem, but also provides a wealth of information about the design space. However, this kind of information is rarely communicated in an effective way to the designer. The result generator enables the user to review this information interactively. For example, the summary window (Figure 16) will always show the latest information from the computational unit. By adding the free software GNUPLOT `<http://www.gnuplot.info>` into the GUI, any design parameter and other useful information can be selected for plotting for inspection. A sample of a created plot can be seen in Figure 18, which displays the history of fitness variation through the evolutionary process that is summarized in Figure 17.

In the following section, this interactive design tool is applied to a simplified pipe design case. Applying this tool to this simplified model will provide a proof of concept and a test bed; it also enables developers to focus on the methodology and keep the effort tractable. Figure 11-18 shows screen shots of the functions that are already implemented.



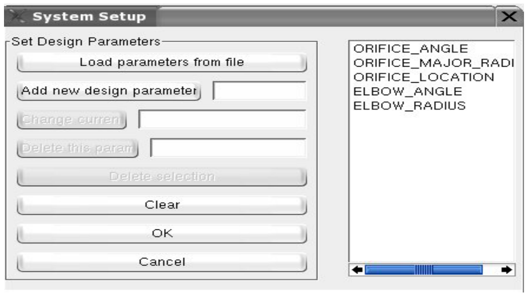
Connected to Executive

Figure 11: MainFrame Provided by VE_Suite



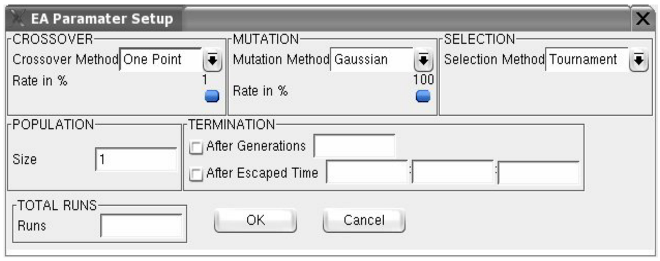
```
Losting the current setup infomationReading
/mnt/rigby/home14/users/hgx/VE_Suite/VE_Suite/demo_test/default.suLosting the current setup infomation
```

Figure 12: Main Window of plug-in GUI



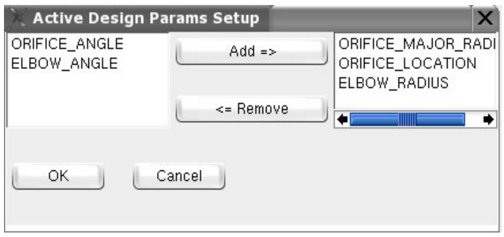
ORIFICE_ANGLE
ORIFICE_MAJOR_RADI
ORIFICE_LOCATION
ELBOW_ANGLE
ELBOW_RADIUS

Figure 13: Design Params Setup Dialog



Crossover Method: One Point; Rate in %: 1
Mutation Method: Gaussian; Rate in %: 100
Selection Method: Tournament
Size: 1
Runs:

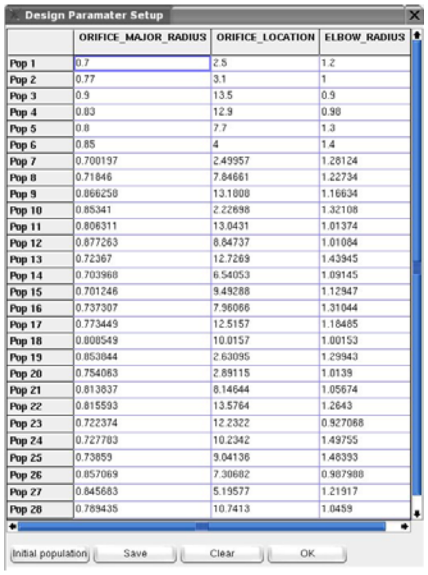
Figure 14: EAs Contraol Params Setup Dialog



ORIFICE_ANGLE
ELBOW_ANGLE

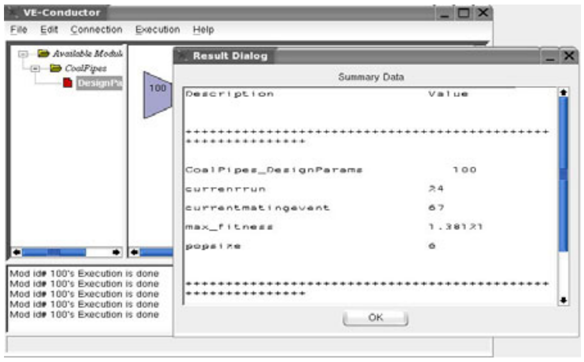
ORIFICE_MAJOR_RADI
ORIFICE_LOCATION
ELBOW_RADIUS

Figure 15: Active Design Params Setup Dialog



	ORIFICE_MAJOR_RADIUS	ORIFICE_LOCATION	ELBOW_RADIUS
Pop 1	0.7	2.5	1.2
Pop 2	0.77	3.1	1
Pop 3	0.9	13.5	0.9
Pop 4	0.83	12.9	0.98
Pop 5	0.8	7.7	1.3
Pop 6	0.85	4	1.4
Pop 7	0.700197	2.49957	1.28124
Pop 8	0.71846	7.84661	1.22734
Pop 9	0.866258	13.1808	1.16634
Pop 10	0.85341	2.22698	1.32108
Pop 11	0.806311	13.0431	1.01374
Pop 12	0.877263	8.84737	1.01084
Pop 13	0.72367	12.7269	1.43945
Pop 14	0.703968	6.54053	1.09145
Pop 15	0.701246	9.49288	1.12947
Pop 16	0.737307	7.96066	1.31044
Pop 17	0.773449	12.5157	1.18485
Pop 18	0.808549	10.0157	1.00153
Pop 19	0.853844	2.63095	1.29943
Pop 20	0.754063	2.89115	1.0139
Pop 21	0.813837	8.14644	1.05674
Pop 22	0.815593	13.5764	1.2643
Pop 23	0.722374	12.2322	0.927068
Pop 24	0.727783	10.2342	1.49755
Pop 25	0.73859	9.04136	1.48393
Pop 26	0.857069	7.30682	0.987988
Pop 27	0.845683	5.19577	1.21917
Pop 28	0.789435	10.7413	1.0459

Figure 16: Initial Population Table



```
Mod id# 100's Execution is done
Mod id# 100's Execution is done
Mod id# 100's Execution is done
Mod id# 100's Execution is done
Mod id# 100's Execution is done
```

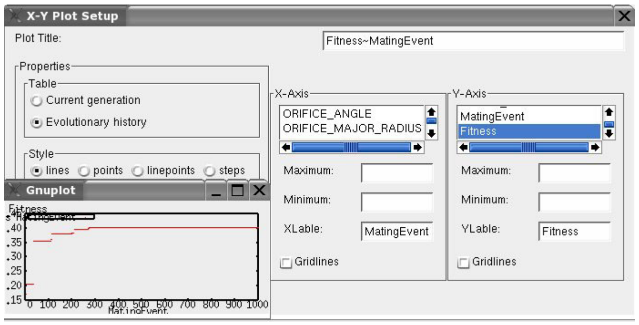
```
Description                              Value

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

CoalPipes_DesignParams                      100
currentrun                               24
currentmatingevent                       67
max_fitness                              1.38121
popsize                                  6

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 17: Example of Result Paramter



Plot Title: Fitness~MatingEvent
Table: Current generation [✓ Evolutionary history]
Style: [✓ lines] points linepoints steps
ORIFICE_ANGLE
ORIFICE_MAJOR_RADIUS
Maximum:
Minimum:
XLable: MatingEvent
MatingEvent
Fitness
Maximum:
Minimum:
YLable: Fitness

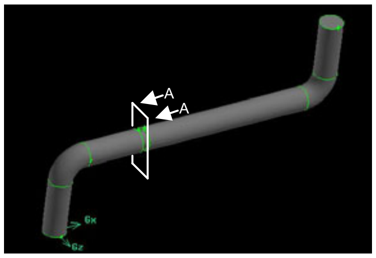
Figure 18: Example of Result Plot

4 TEST CASE

The optimization problem used here as a test case is a simplified coal transport pipe which operates under the dilute phase condition. Currently, dilute phase pneumatic conveying system design is largely dependent on the use of integral modelling techniques such as head loss coefficients. In general cases, as long as the head loss requirement is met, the details of gas-solid flow, such as particle distribution and velocity profile are not used to drive the search process. It is well-known that in order to achieve constant high performance from the furnace, the coal transport piping system should be designed such as to ensure uniform rate and continuous feed of pulverized fuel to the burners. Therefore, in the case of coal piping system of a coal-fired power plant, more information than the head loss and flow rate is needed to develop high-quality designs. To meet the operation requirement, a uniform coal distribution across pipe cross-sections is necessary; this type of information can be obtained by detailed multiphase CFD analysis.

One of the most important factors contributing to uneven coal distribution is the development of coal roping. Coal roping is generally found in elbows and is hard to prevent. The details of mechanisms that result in coal roping can be found in Schallert (2000), Schneider (2002) and Yilmaz (1997)

Figure 19 shows the geometry of a test pipe. This basic pipe geometry consisted of a vertical pipe with a length of five pipe diameters, the elbow section, and a horizontal pipe with a length of fifteen pipe diameters followed by another elbow and a vertical pipe with a length of three pipe diameters. Both elbows are 90˚ elbows and the radius of the elbows is three times the pipe diameter. In this design problem, one elliptical orifice is installed somewhere in the long horizontal pipe to break the coal rope. The lower part of Figure 19 shows the parameters determining the shape of the orifice. The design parameters in this case



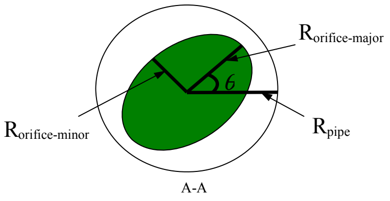


Figure 19: Sketch of Testing Pipe Geometry

Table 1 : Geometry Sizes of the Testing Facility

Pipe Diameter, d (m)	First Vertical Pipe Length	Horizontal Pipe Length	Second Vertical Pipe Length	Elbow Radius
d =0.154	$5\times d$	$15\times d$	$3\times d$	$3\times d$

include the location of the orifice, L, the orientation of the orifice, θ, and the ratio of the radius of orifice ($R_{orific\text{-}major}$) and the pipe, $\tilde{r}$. The goal of the pipe design optimization is to minimize the coal distribution difference at the exit surface of the pipe. To facilitate a quantitative comparison between the cases with different orifices, a mixing index (*MI*) defined by Bilirgen (1998) is used as the index to the coal distribution. The mixing index is computed using:

$$MI = \frac{1}{C_p}\left[\frac{1}{n-1}\sum_{k=1}^{n}\left(C_m(k)-C_p\right)^2\right]^{1/2} \quad (1)$$

Where C_p is the mean particle concentration in the pipe's cross-section, $C_m(k)$ is the local particle concentration measured at different locations in the x-direction, and n is the total number of measurements along the pipe diameter. Equation 1 gives the degree of variation in concentration as a standard deviation. The flow is assumed to be well mixed when mixing index tends to zero. The optimization problem is then formulated as shown in Figure 20. In Figure 20 $S_{orifice}$ and S_{pipe} represent the area of the cross section of orifice and pipe, respectively.

Minimize:

$$MI$$

where:

$$0^o < \theta < 90^o$$
$$2\cdot d < L < 14\cdot d$$
$$0.7 < \tilde{r} < 0.9$$
$$\frac{S_{orifice}}{S_{pipe}} = 0.8$$

Figure 20: Test Case Problem Setup

In this study, Gambit™, which is the preprocessor of the Fluent™ package, is used to build geometry based on parameters generated by computers or chosen by users. The renormalization group RNG $k-\varepsilon$ turbulence model provided by the Fluent™ was used to simulate turbulent gas-particle flows through the pipe. A C program written with the user-defined function (UDF) API provided by Fluent™ is used to calculate the mixing coefficient. The Evolve function in the Model class (see section 2) links above pipe simulation with the evolutionary algorithm so that proposed design changes are automatically evaluated. Figure 21 shows a simplified block diagram of the Evolve function. As shown, the process proceeds in an iterative manner; for each feasible design change, a CFD analysis is automatically executed in the background to evaluate fitness. The CFD analysis is followed by another design change along with the computational re-meshing of the new design. The design is again evaluated, and through the evolutionary optimization process, the best design is found.

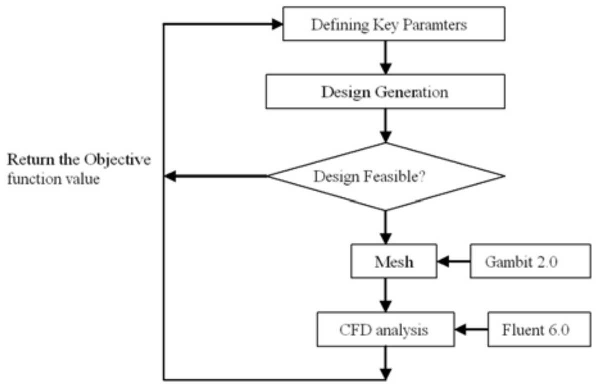


Figure 21: Evolve Function

It is well known that choosing an initial population is important when using EAs as a searching method. A good choice can result in rapid convergence, whereas a bad choice can cause the algorithm to search randomly for a very long time before closing in on the solution. Selecting the initial design population is very straightforward in our system. After testing different design parameters, the effect of the key active parameters upon the design problem can be studied and evaluated. As a result, the users can interactively explore the design candidates and also gain additional insight into the underlying physical and mathematical models for the proposed designs, which in turn aids the optimization of the configuration and geometry of the design. Moreover, users' experience and knowledge are integrated into this exploration process and helps them choose the initial designs that are relatively close to the optimum result. It may take some extra time to find these relatively optimal starting design configurations because users have to test many cases before they can determine which proposed designs should be used as initial designs. The virtual environment provides a natural interface between humans and computers; it helps users grasp information quickly and accurately, which has been demonstrated in many VR-related publications.

Three runs were carried out in this study; each has an initial population of 32 and stopped after 770 generations. In the first two runs, the members of the initial population were selected arbitrarily from the design space by the computer. In the third run, designers defined the first 24 members (from 58 tries) based on their experience gained from the interactive process; the computer randomly generated the remaining 8 members to keep diversity amongst the population. With all three runs, the computer randomly generated the crossover operator probability p_c and the mutation operator probability p_m. The technique successfully converged; the optimum parameters such as orifice radius and orifice angles, converged to very close values with different starting initial design candidates. The variations of the best individual's fitness during the evolution process in the run are shown in Figure 22. It shows that with a good initial population, EAs can find the optimal solution much faster. In our case, the required mating events reduces from 650 to 330. Table 2 shows that the computational time was reduced to 96 hours from the original 175 hours by using the new interactive design platform.

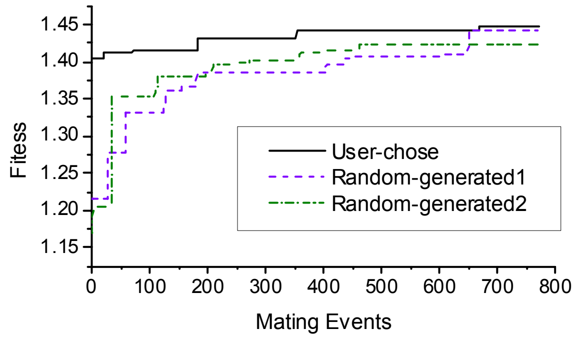


Figure 22: Fitness Variations in the Evolution Processes

Table 2: Comparison between Traditional and Interactive Design System

	Traditional Design System	Interactive Design System
Total Popsize	32	32
Randomly-Generated Initial Popsize	32	8
User-chosen Initial Popsize	0	58
Mating Events	650	330
Time Required to Complete One Case by Computer (min)	4	4
Average Time Spent by Users for Choosing One Design Candidate (min)	0	5
Total Calls to CFD solver	2632	1378
Total Time (h)	175	96

While the above case shows the value that can be gained from our new interactive design system, users can also freely monitor the health of this search process by checking earlier results. For instance, in this coal pipe case, if the constraint on the orifice area is forgotten, the optimizer would try to reduce the orifice area to increase mixing capability of the turbulence. Therefore, after several iterations, the design is driven towards physically unreasonable space. Most likely, users would detect this error in the design formulation earlier by checking the current status of the design space in the virtual environment. For a more complicated product design, these wasted iterations could be time-consuming and costly. Keeping users in the loop allows them to detect a flaw in the mathematical model, and change the optimization early, thus decreasing the amount of time wasted on generating incorrect results.

5 CONCLUSION AND FUTURE WORK

In this paper, we present a novel method for building a new interactive design. A framework was established for adding human interaction into the design loop; this is a big step forward in the development of an interactive design system. This system provides an easy and intuitive way to introduce designers into the optimization process. As demonstrated in the test case, we were able to reduce the total computational time for the test case by more than 40%. Hence, it is safe to say, for a simple CFD related design optimization problem, this virtual engineering design tool can help designers solve the problem within a much shorter time frame. Additional work needs to be focused on parallelizing the CFD optimization tasks since it is widely known that EAs can be used to take full advantage of a parallel computer structure.

REFERENCES

Anderson D., E. Anderson, N. Lesh, J. Marks, B. Mirtich, D. Ratajczak, and K. Ryall. 2000. Human-guided simple search. In *Proceeding Of AIAA 2000*.

Bilirgen, H., E. Levy, and A. Yilmaz. 1998. Prediction of pneumatic conveying flow phenomena using commercial CFD software. In *Powder Technology*, 95:37-41.

CORBA, Available via Object Management Group `<http://www.omg.org/>` [accessed April 12, 2005].

Goldberg, D. E. 1998. *Genetic algorithm in search, optimization and machine learning*. Addison-Wesley, MD.

Hong L., T. Mingxi, and H. F. John. 2004. Supporting creative design in a visual evolutionary computing environment. In *Journal Advances in Engineering Software*, 35:261-271.

McCorkle, D.S., K. M. Bryden, and S. J. Kirstukas. 2003. Building a foundation for power plant virtual engineering. In *Proceedings of the 28th International Technical Conference on Coal Utilization and Fuel Systems*, 118-127, Clearwater, FL.

Schallert, R. and E. Levy. 1998. Effect of a combination of two elbows on particle roping in pneumatic conveying. In *Power Technology*, 107:226-233.

Schneider, H., T. Frank, D.K. Pachler, and K. Bernert. 2002. A numerical study of the gas-particle flow in pipework and flow splitting devices of coal-fired power plant. In *10th Workshop on Two-Phase Flow Predictions*, 227-236.

SCIRun, Available via `<www.sci.utah.edu>` [accessed April 12, 2005].

SGI-OpenGL Performer, Available via `<www.sgi.com/products/software/performer>` [accessed April 12, 2005].

TAO CORBA, Available via `<www.ece.uci.edu./~schmidt/TAO.html>` [accessed April 12, 2005].

The Visualization ToolKit (VTK), Available via `<http://public.kitware.com/VTK>` [accessed April 12, 2005].

Vladimir O. B, C. Christophe, G. Dipankar, Q. Gary, N. V. Garret. 2002. VisualDOC: a software system for general-purpose integraton and design optimization. In *AIAA/ISSMO'2002*, Atlanta-Georgia.

VRJuggler, Available via `< http://www.vrjuggler.org >` [accessed April 12, 2005].

wxWidgets, Available via `<http://www.wxWindows.org>` [accessed April 12, 2005].

Xiao, A. and K. M. Bryden. 2004. Virtual Engineering: A vision of the next-generation product realization using virtual reality technologies. In *Proceedings of ASME Design Engineering Technical, Computers in Engineering Conference, Salt lake City, Utah, DETC2004/CIE-57698.*

Yilmaz, Ali. 1997. Roping phenomena in lean phase pneumatic conveying. *Ph.D. Thesis*, Lehigh University.

AUTHOR BIOGRAPHIES

GENGXUN HUANG is a doctoral candidate in the Department of Mechanical Engineering at Iowa State University. She is a member of ASME and AIAA. Her e-mail address is `<hgx@iastate.edu>`

KENNETH M. BRYDEN is an associate professor and associate chair of the Department of Mechanical Engineering at Iowa State University. His e-mail address is `<kmbryden@iastate.edu>`