

# White Paper: Using the HyperSizer Object Model for Software Integration

Phil Yarrington, Craig Collier, Mark Pickenheim  
Collier Research Corporation  
December 2001

## Introduction

A capability is now included in HyperSizer that allows it to be called from many other software products. This new capability, called the Object Model, was developed to address HyperSizer's inability to operate without its graphical user interface in a non-interactive or heterogeneous network environment.

The HyperSizer Object Model is built on COM and ActiveX, the core technologies of Microsoft Windows, on which nearly all Windows programs are built. This means that it integrates very smoothly with Windows programs. In addition, distributed computing technologies are readily available (DCOM, Java RMI, Enterprise Java Beans, CORBA, ModelCenter, iSIGHT) which expose the HyperSizer Object Model over heterogeneous networks that include UNIX, Linux, and Windows workstations.

The HyperSizer Object Model has successfully been integrated with Microsoft Excel (actually works with any Microsoft Office product), Mathcad, Microsoft Visual Basic, and Microsoft C++. HyperSizer analyses were successfully performed from SGI Unix workstations using Java and its built-in Remote Method Invocation (RMI). One of the key motivations for the development of the Object Model is that it enables HyperSizer to become part of a large multi-disciplinary design system, as shown in Figure 1. NASA successfully integrated HyperSizer into its own heterogeneous, multidisciplinary batch design system, called the Environment for Launch Vehicle Synthesis (ELVIS), by calling HyperSizer from Phoenix Integration's Analysis Server and ModelCenter software.

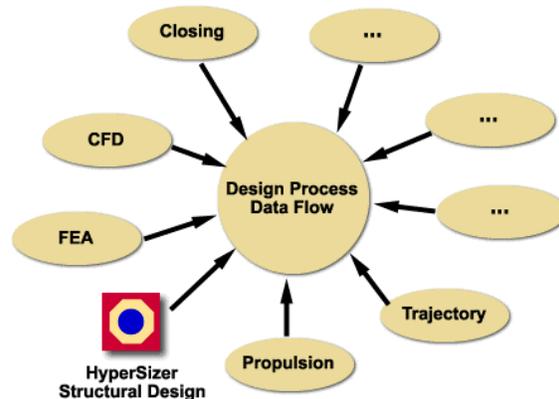


Figure 1: The HyperSizer Object Model enables HyperSizer to become part of a large multi-disciplinary analysis and design system

## **Object Model Overview**

In the past, HyperSizer ran strictly as an interactive program. Two motivations led to the development of the Object Model. First was the inability of HyperSizer to be called in a non-interactive, batch environment. The second related motivation was to allow HyperSizer to be integrated into a large design process on a heterogeneous network of computers. A traditional approach to these problems might be to interface codes by passing input and output ASCII files from one process to another and tie them together with a scripting language. We do not view this approach as commercially dependable due to lack of robustness and maintainability, two factors that are especially important because HyperSizer is being used by NASA and industry for analysis and design of next generation structures. Our preferred approach is to integrate HyperSizer using distributed objects or web based solutions, rather than relying on passing and parsing of data files.

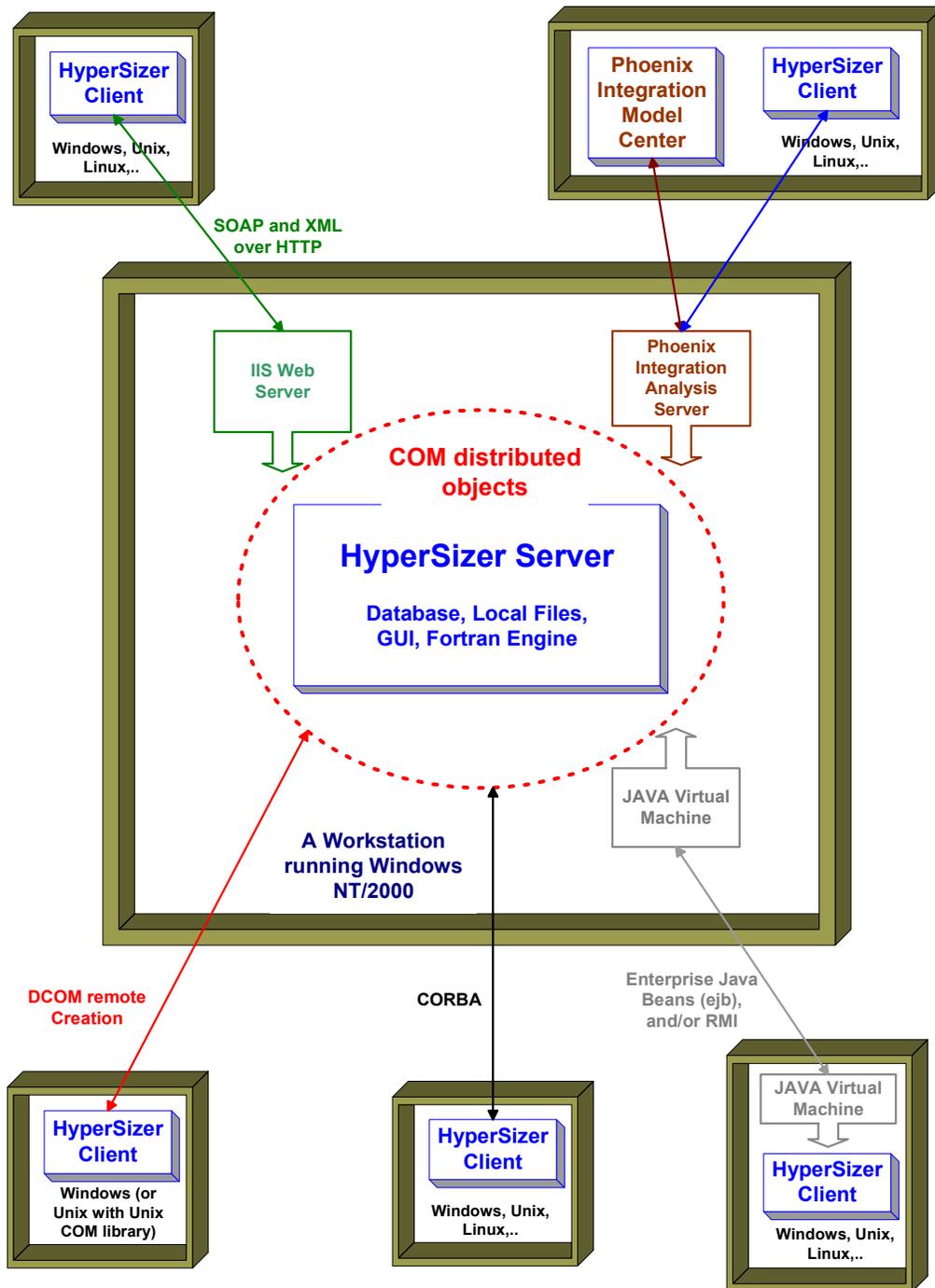
HyperSizer has always been a native Microsoft Windows application built on *Microsoft COM* and *ActiveX* technology. COM stands for Component Object Model and is the object model on which all native Windows applications, including the operating system itself, is based. ActiveX, formerly known as OLE (or Object Linking and Embedding), is the technology that enables one process or object to use functions or properties from another process or object. For example, ActiveX enables spreadsheets developed using MS Excel to be embedded into MS Word documents.

NASA and other customers have successfully integrated our software with:

- Unix and Linux Workstations
- Excel (MS Office)
- Visual Basic, C++
- Java
- Mathcad
- ModelCenter / Analysis Server

Because HyperSizer is built on ActiveX, much of its functionality has been exposed to outside processes as an *ActiveX Automation Server*. This means that client programs, built using any COM aware programming language, can instantiate (or create) objects from this server and ask these objects to perform functions. For example, a Java applet can open a HyperSizer database (using a HyperSizer **Application** object), retrieve a list of HyperSizer projects, and export the materials used by a particular project. The same Java applet could automatically size a structural component using a **Component** object. The HyperSizer server is intended to provide batch functions that can be executed repetitively without user intervention.

With the proper license, the HyperSizer Object Model is available on any computer where HyperSizer is installed. It can be referenced by any COM aware application (e.g. MS Excel) or programming language (e.g. Visual Basic, Java, C++). Once installed on a Windows computer, the Object Model can be accessed using any of the Client-Server technologies shown in Figure 2, which illustrates the various ways the Object Model is exposed to a heterogeneous environment of mixed computing platforms (Windows, Unix, etc).



Collier Research Corp. 1-16-01

Figure 2: The HyperSizer Object Model exposes HyperSizer to a heterogeneous network of computers with a variety of connection protocols. In the figure, each box represents a different workstation, some running Windows XP/2000/NT and some running UNIX or Linux. The connections between the platforms represent various connection protocols such as SOAP-XML, JAVA, etc.

### Object Model Description

A simplified graphical version of the HyperSizer Object Model is shown in Figure 3. The process begins by creating an instance of an **Application** object. This is the only object that can be instantiated independently, i.e. not through another HyperSizer object. Once the Application object is instantiated, all other HyperSizer objects are instantiated through methods or properties of the Application object or from methods or properties of one of its children. Once an Application object is created, the next step is to get a reference to a **Project** object using the *Projects* property of the Application object (this property is written as 'Application.Projects'). The Project object can refer to an existing HyperSizer project, or a newly created one. Using this Project object, the calling program can modify, size and/or extract data from the project or use the object to get references to **Group**, **Assembly** or **Component** Objects. Each of these objects can, in turn, be used to modify, size and/or extract data from various pieces of the structure.

Through the Object Model, most entities can be:

- Created or deleted
- Queried for properties or results
- Updated with new properties
- Analyzed / Sized

Available Objects Include:

- Projects
- Assemblies
- Optimization Groups
- Structural Components
- Load Sets / Load Cases
- Materials

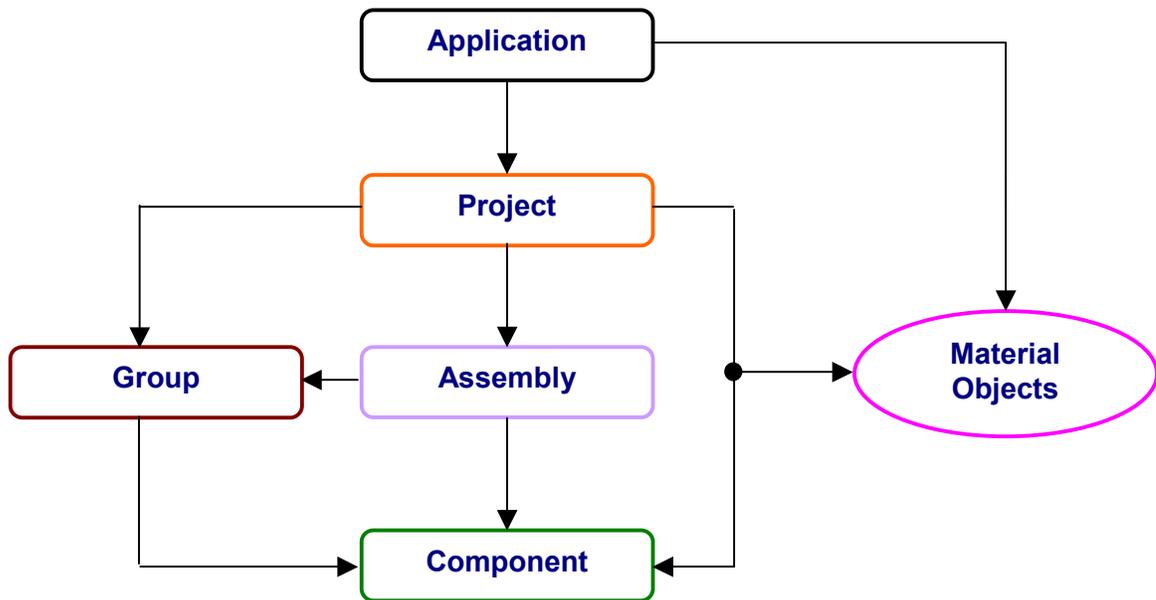


Figure 3: Simplified HyperSizer Object Model

A typical program for modifying the buckling lengths of a series of HyperSizer components in a project called "Ap1 Demo", re-sizing them, and getting the Margins of Safety, might look like the following:

Start Program

Set variable AppObject = new Application Object

Set variable ProjectObject = Project object created from Projects property of AppObject (referring to the project named "Ap1Demo")

Set variable GroupObject = Group object created from the Groups property of the Project object

Set variable Components() = Collection of component objects created from Components property of GroupObject

**For Each** ComponentObject in the Components() collection:

Set ComponentObject.PanelProperty("BucklingLength") = New Buckling Length  
Save the new buckling length to the database (GroupObject.Save)  
Size the component (ComponentObject.Size)  
Set variable MOS = ComponentObject.PanelResult("Minimum MOS")  
Output (to screen, file, etc.) ComponentObject.Name, MOS

**Next** ComponentObject

Release the object variables Components, GroupObject, ProjectObject, AppObject

End Program

The execution of this pseudo-code creates an Application object, then uses this object to get a Project object, which is used to get a Group object. The Group object in turn gets a collection of Component objects. Then for each component object, the buckling length is modified, the component is resized, and the margin of safety is reported.

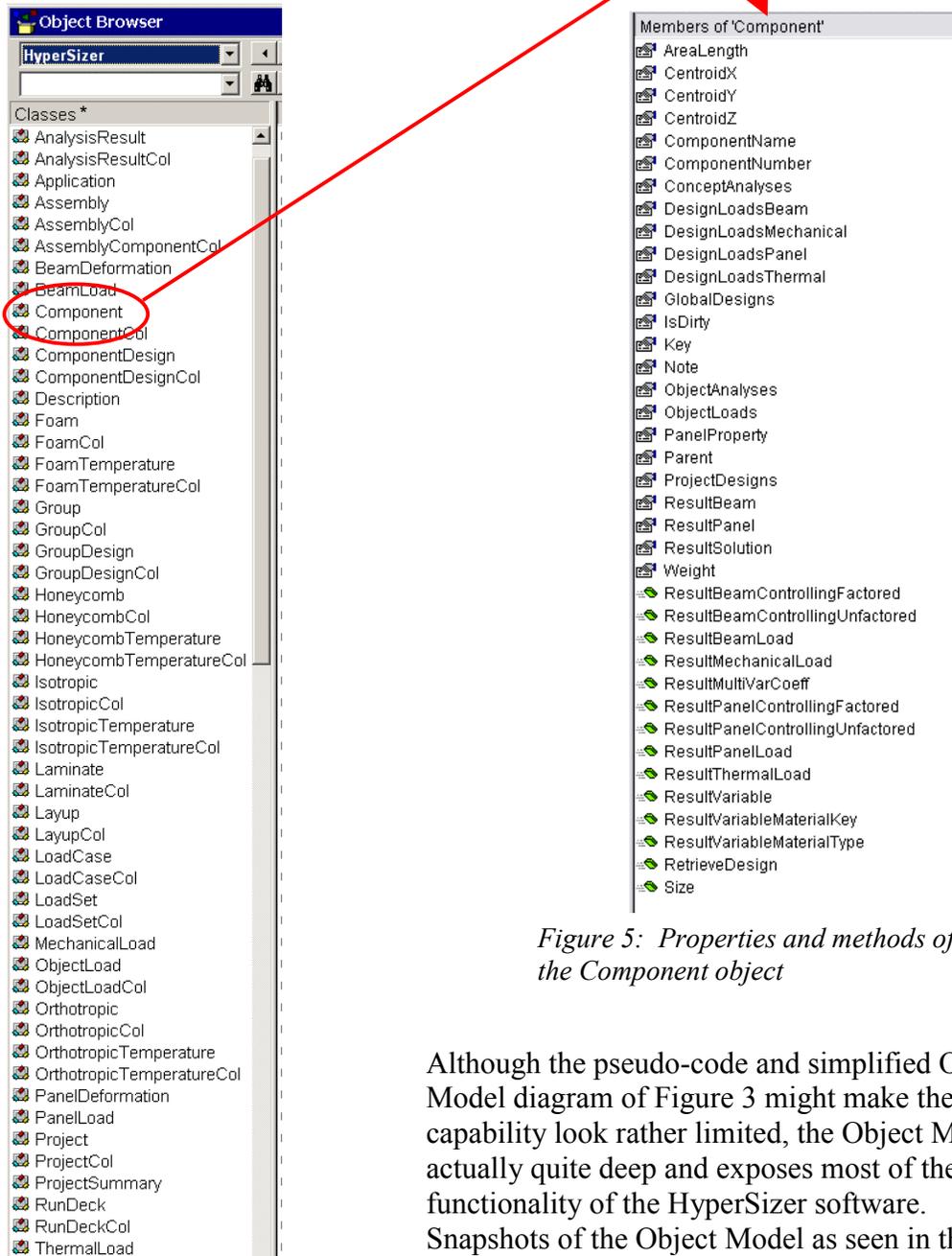


Figure 5: Properties and methods of the Component object

Figure 4: Objects available through the HyperSizer Object Model (\*A class is an object oriented programming concept which refers to a template or blueprint for creating an object)

Although the pseudo-code and simplified Object Model diagram of Figure 3 might make the capability look rather limited, the Object Model is actually quite deep and exposes most of the functionality of the HyperSizer software. Snapshots of the Object Model as seen in the “Object Browser” of the Microsoft Excel Visual Basic editor are shown in Figures 4 and 5. Each item in the Figure 4 list is an object that can be instantiated through the object model. Each object has multiple properties and methods that are used to modify data, size components, groups, assemblies, and projects and extract results. For example, the methods and properties of the Component object are shown in Figure 5.

## Object Model Examples

Several examples of using the Object Model are described in the following pages. In the first example, it is used to solve a global beam spacing optimization problem, which demonstrates the functionality added to the standard interactive HyperSizer. The optimization example illustrates how the HyperSizer Object Model is called from three different software packages, Excel, MathCAD, and Analysis Server/ModelCenter. Next, Java is used to call HyperSizer and create and size an entire project. Finally, a demo created for the NASA Environment for Launch Vehicle Synthesis (ELVIS) shows how HyperSizer fits into a larger design system.

## Frame Spacing Optimization

### Problem Description

In this first example, a curved hat-stiffened panel, representative of a typical SLI/RLV cylindrical fuselage section is subjected to a compressive axial load. J-Beam ringframes are attached to the skin with a given frame spacing as illustrated in Figure 6. Given the ringframe spacing and associated axial loads, the interactive HyperSizer product is able to analyze and optimize the detail design (facesheet, flange and web thickness, hat spacing, materials, etc.) of the frames and the stiffened panel separately. However, it is not aware of the relationship between the weight of the combined structure and the distance between the ringframes. The sizing of the stiffened panel is principally controlled by longitudinal panel buckling, which means that the panel weight increases as the buckling length increases (i.e. the ringframe spacing increases). Therefore, the optimum weight of the stiffened panel alone is obtained by minimizing the ringframe spacing. However, as the ringframe spacing decreases, the number of frames per unit

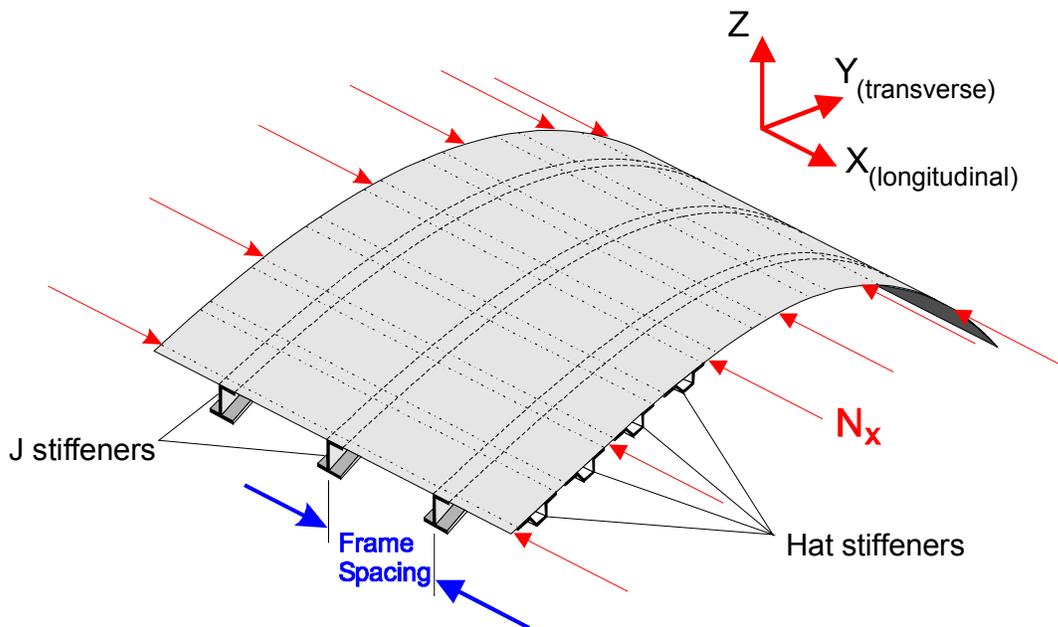


Figure 6: SLI/RLV Fuselage frame spacing problem

longitudinal length increases, and the weight for the J-Beams goes up. The following example applications call HyperSizer to size the stiffened skin and ringframes in an iterative process to arrive at an optimum system weight. They are used to illustrate several ways of interfacing with the HyperSizer Object Model. First, the optimization is solved using the non-linear “solver” capability of Microsoft Excel. Second, MathCAD is integrated with the Object Model to document the equations used by this example. Finally, the optimization is solved using the built-in optimizer of Phoenix Integration’s ModelCenter.

Excel non-linear solver implementation

A spreadsheet was built that uses Excel’s non-linear solver to optimize the beam spacing. The spreadsheet has an underlying Macro code written in Visual Basic for Applications (VBA) that interfaces with the HyperSizer Object Model. A snapshot of the spreadsheet with a snippet of the VBA code is shown in Figure 7. In the non-linear solver, the cell targeted for minimization is E37 (the total system unit weight) and the only cell changed by the solver is B30 (the frame spacing). As the solver changes the frame spacing, the beam and panel unit weights (column C) are solved by calling the HyperSizer Object Model through the VBA macro. The spreadsheet then smears the ringframe weights (which are in pounds per foot) into area unit weights and calculates the total system unit

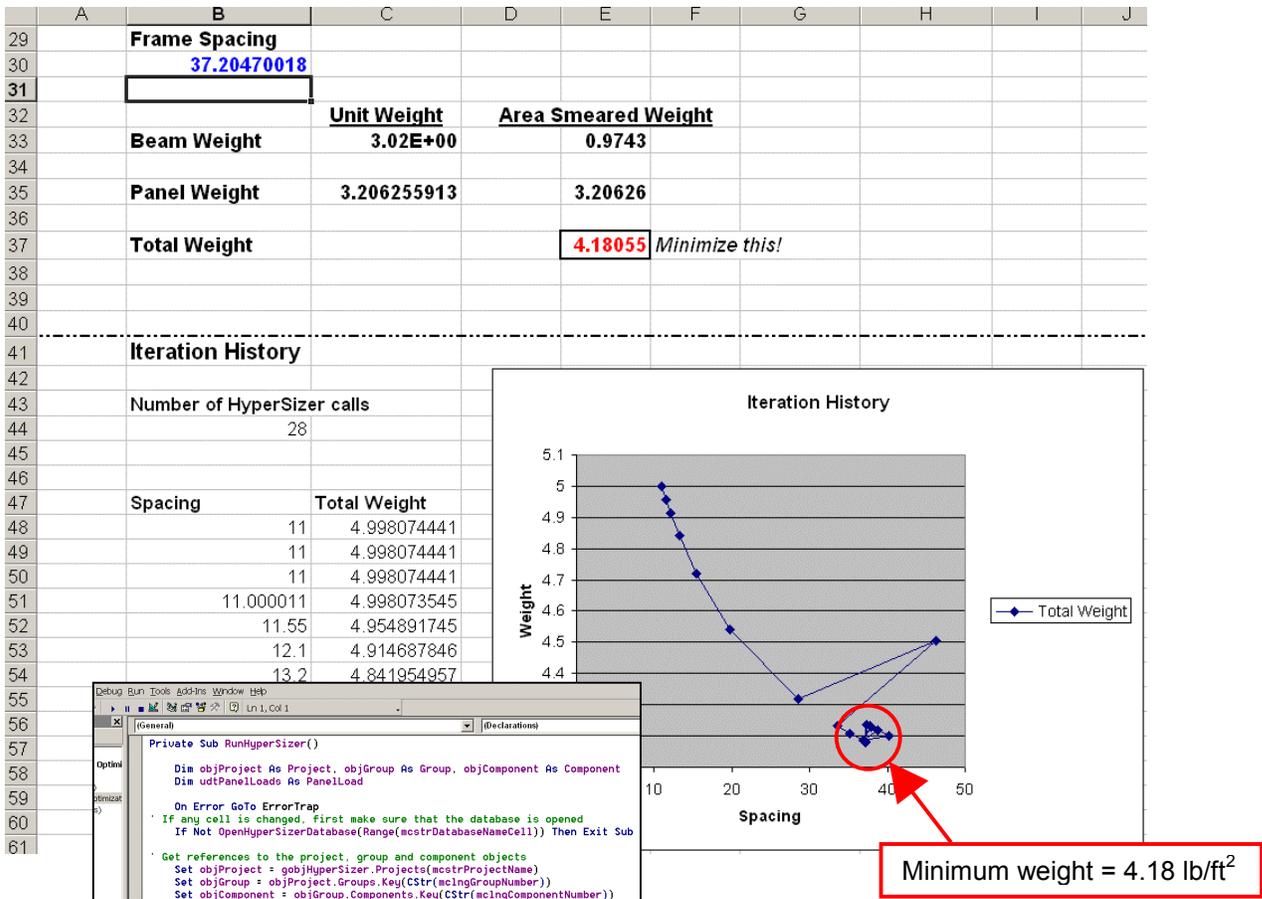


Figure 7: Excel spreadsheet optimization of the SLI/RLV ringframe spacing

weight. As the solver changes the frame spacing, the unit weight for the current iteration is added to the “Iteration History” graph in the spreadsheet. The graph of iteration history in Figure 7 shows how the software quickly iterates to an optimum frame spacing for this particular loading scenario of 37.2 inches with a corresponding system unit weight of 4.18 pounds per square foot.

Implementing HyperSizer’s panel optimization is relatively straightforward. Sizing of the J-Beam ringframes is discussed in the next section as the Object Model is called from another software product, MathCAD.

### MathCAD implementation

Because the loads in the fuselage are orthogonal to the J-Beam ringframes, the actual load in these beams is relatively low. However, a danger of under designing these beams is that general instability of the fuselage can occur as depicted in the bottom image of Figure 8. Michael Niu’s book on aircraft design [1] has a very simplistic method for determining the beam stiffness required to prevent general instability.

The equations from [1] were implemented in a MathCAD document as shown in Figure 9. Niu’s equations yield a total equivalent stiffness (EI) that is required for the ringframes. In the bottom part of the MathCAD document, the equivalent stiffness calculated from this equation, along with the frame spacing, are sent to the Object Model to solve for panel and beam weights.

This MathCAD document is not an optimization of the ringframe spacing, but rather documents the panel and beam local sizing for a given spacing. It demonstrates how HyperSizer can be called from MathCAD in a flexible environment for creating documentation and stress reports.

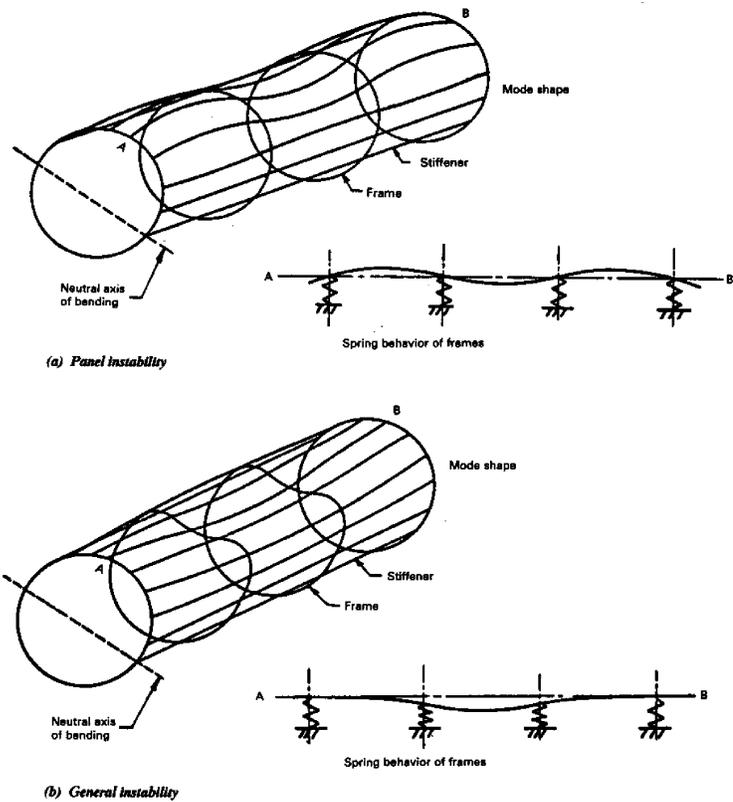


Figure 8: J-Beam ringframes are sized to prevent fuselage general instability (from Ref. [1])

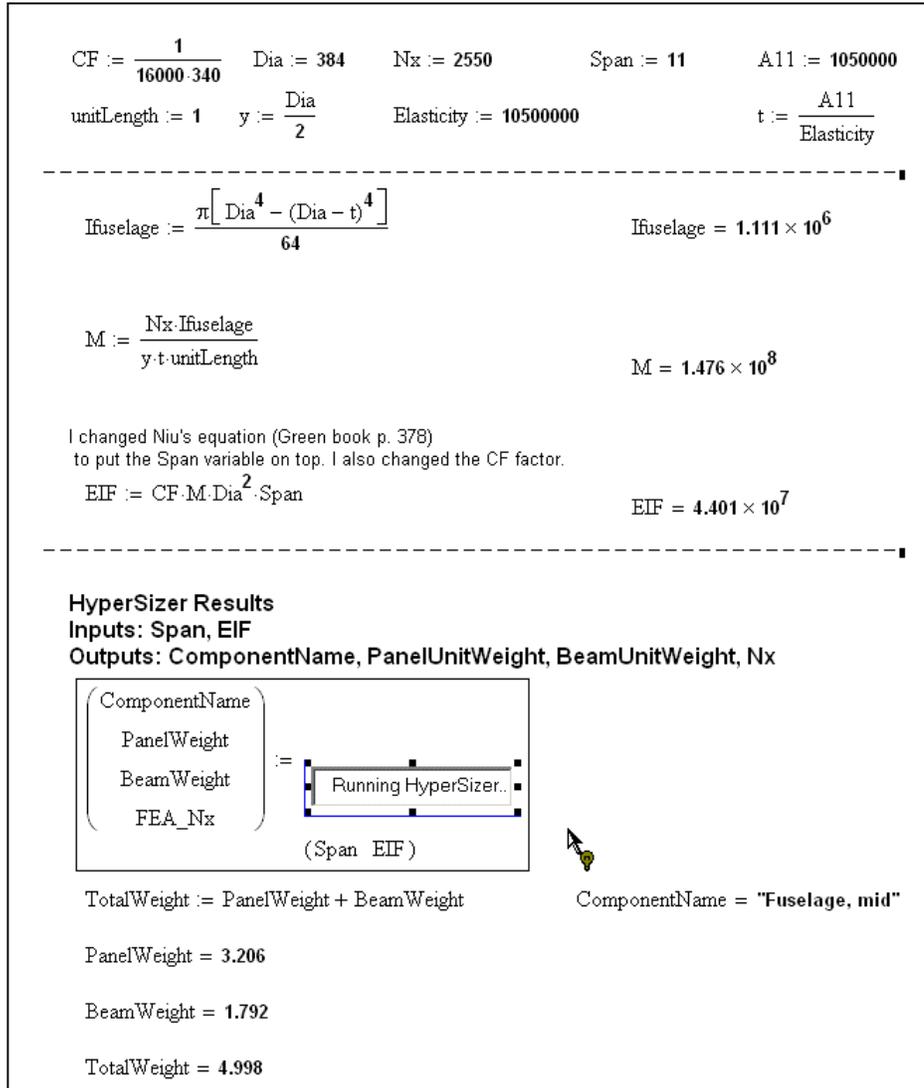


Figure 9: Mathcad implementation of stiffened skin and ringframe sizing

ModelCenter Implementation

As a final example using the fuselage ringframe spacing example, the same problem is solved using Analysis Server and ModelCenter from Phoenix Integration. This solution is essentially the same as the Excel non-linear solver solution described earlier, except that in this case, a ModelCenter Optimizer component was used to minimize the system weight.

The first step is to build Analysis Server wrappers that expose HyperSizer Object Model to ModelCenter. Analysis Server wrappers are ASCII text files that describe the interface

of a program (like HyperSizer) to the Analysis Server application. Once the wrappers were built and Analysis Server running, they were plugged into ModelCenter and tied together with an “Optimizer” component. Two wrappers for the Object Model were made, called HyperSizerPanel and HyperSizerBeam, which are used to size the stiffened panel and the J-Beam ringframes respectively. The ModelCenter optimizer iteratively calls these two wrappers and sums up the resulting unit weights to get an overall system unit weight. The convergence on an optimum spacing can be seen in the attached “Data Collector” window.

One of the features of Analysis Server that makes this a powerful solution, is that once the HyperSizer Object Model is exposed through an Analysis Server wrapper, it is automatically available not only to ModelCenter on the machine where HyperSizer is installed, but also from other platforms on the same network, including Windows and Unix workstations.

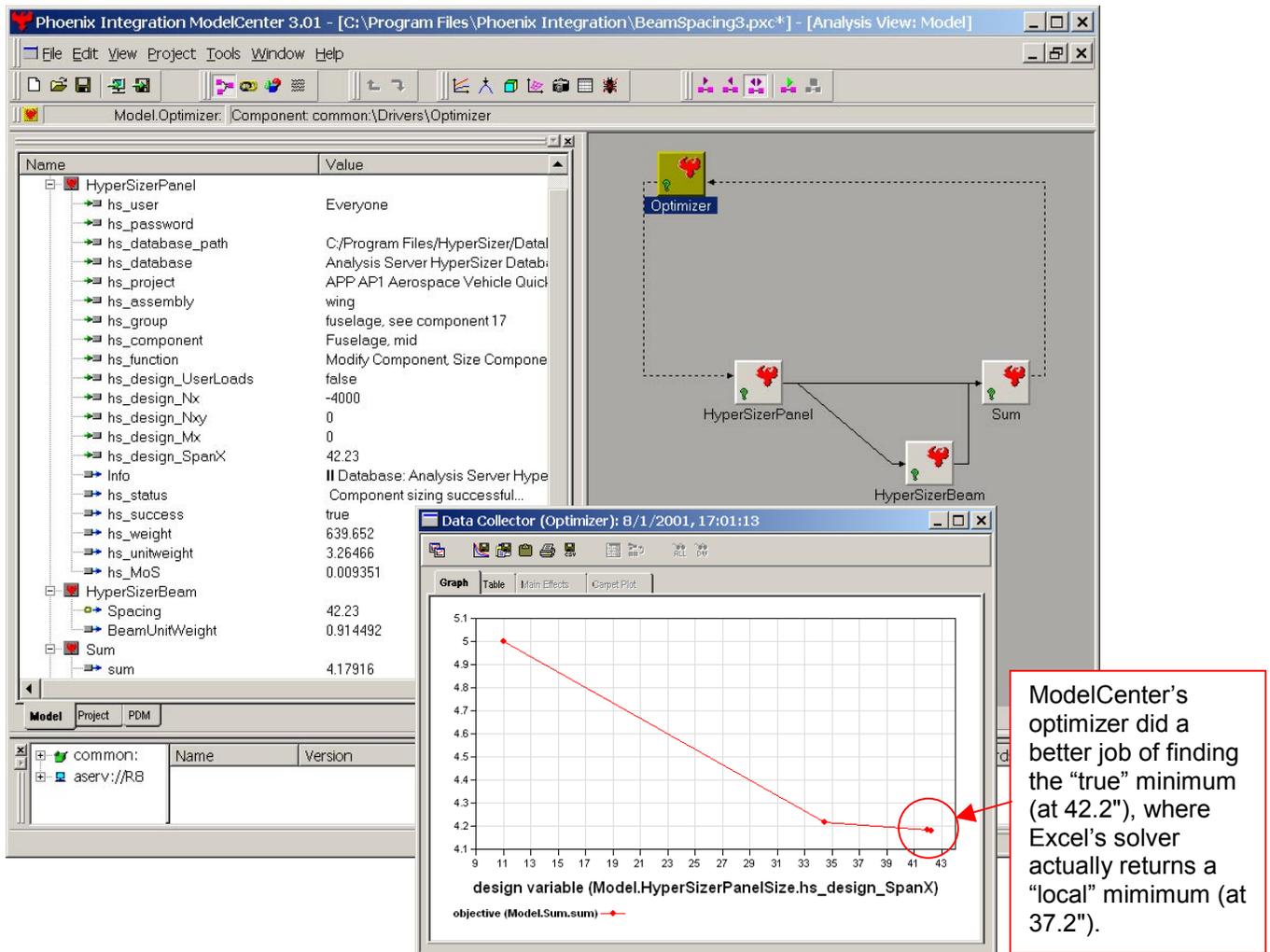
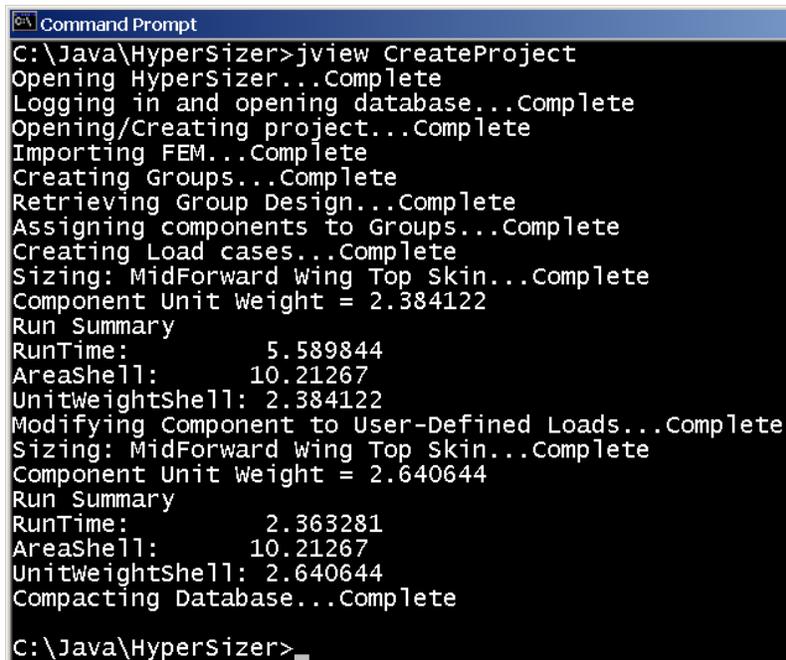


Figure 10: ModelCenter optimization of the SLI/RLV ringframe spacing

## HyperSizer Object Model called from Java

The Object Model can be called from Microsoft COM aware applications such as Excel, MathCAD and ModelCenter, but it can also be called from many modern programming languages such as Visual Basic, C++ and Java. As an example, a Java program was built that creates and analyzes a HyperSizer project from scratch. Figure 11 shows a snapshot of the output from the Java program. The significance of this example is that it demonstrates how a HyperSizer project can be built and analyzed from the beginning including import of a finite element model, assignment of structural components to optimization groups, specification of optimization parameters and assigning user-defined loads without user intervention. This type of automation makes it possible to include HyperSizer in a large multi-disciplinary design system where the overall design is repeatedly changed and must be re-analyzed and re-sized by HyperSizer in automated batch sessions.



```
Command Prompt
C:\Java\HyperSizer>jview CreateProject
Opening HyperSizer...Complete
Logging in and opening database...Complete
Opening/Creating project...Complete
Importing FEM...Complete
Creating Groups...Complete
Retrieving Group Design...Complete
Assigning components to Groups...Complete
Creating Load cases...Complete
Sizing: MidForward Wing Top Skin...Complete
Component Unit Weight = 2.384122
Run Summary
RunTime:          5.589844
AreaShell:       10.21267
UnitweightShell: 2.384122
Modifying Component to User-Defined Loads...Complete
Sizing: MidForward Wing Top Skin...Complete
Component Unit Weight = 2.640644
Run Summary
RunTime:          2.363281
AreaShell:       10.21267
UnitweightShell: 2.640644
Compacting Database...Complete
C:\Java\HyperSizer>
```

Figure 11: Complete project creation and sizing using the HyperSizer Object Model and Java

## HyperSizer ELVIS Demo

In the final example, the Object Model was used to integrate HyperSizer into the Environment for Launch Vehicle Synthesis (ELVIS) Development program at NASA Langley\*. ELVIS uses Analysis Server and ModelCenter to integrate codes together and enable multi-disciplinary analyses to take place across a heterogeneous network of mixed UNIX-Windows computers. HyperSizer was integrated into the Advanced Structures subtask of ELVIS, with the focus of quickly analyzing and predicting structural weights. This subtask integrates a NASA vehicle sizing code called CONSI<sub>Z</sub>, SDR<sub>C</sub> I-DEAS® for CAD and FEA, and HyperSizer for detail design.

A ModelCenter component with the name “HyperSizer” was developed, which accesses the HyperSizer Object Model and fits directly into the larger Advanced Structures ModelCenter model. This component is further broken down into three sub-components, as shown in Figure 12. As the Advanced Structures model runs, a finite element model is built and solved automatically in I-DEAS to get internal load paths. The first HyperSizer sub-component, called “ProjectSetup,” creates a new HyperSizer project called “ELVIS Demo”, imports this I-DEAS finite element model and sets up the HyperSizer load cases. The second sub-component, called “GroupSetup,” creates optimization groups in the new project, assigns optimization parameters and bounds, and assigns structural components to those groups. The final sub-component, “Size,” sizes all of the structural components in the model and returns the overall structural weight and the weight breakdown by

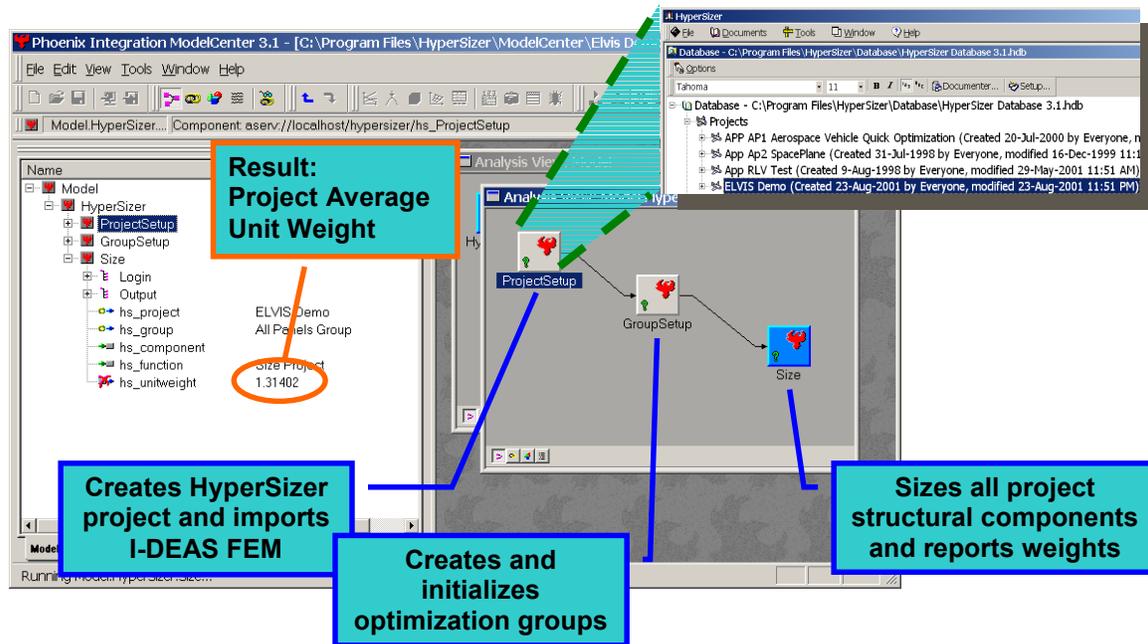


Figure 12: The HyperSizer ModelCenter component creates a new HyperSizer project, imports a finite element model sets up optimization groups and sizes all structural components

\* ELVIS development has been funded by the NASA High Performance Computing and Communications Program (HPCCP)

structural component back to the Advanced Structures model.

During execution of the HyperSizer ModelCenter component, HyperSizer automatically determines controlling failure modes, margins of safety, optimum unit weights, etc. for all structural components based on the internal loads generated from the I-DEAS Finite Element Analysis. The HyperSizer snapshot shown in Figure 13 shows the controlling failure analyses on a full finite element model resulting from execution of the HyperSizer ModelCenter component.

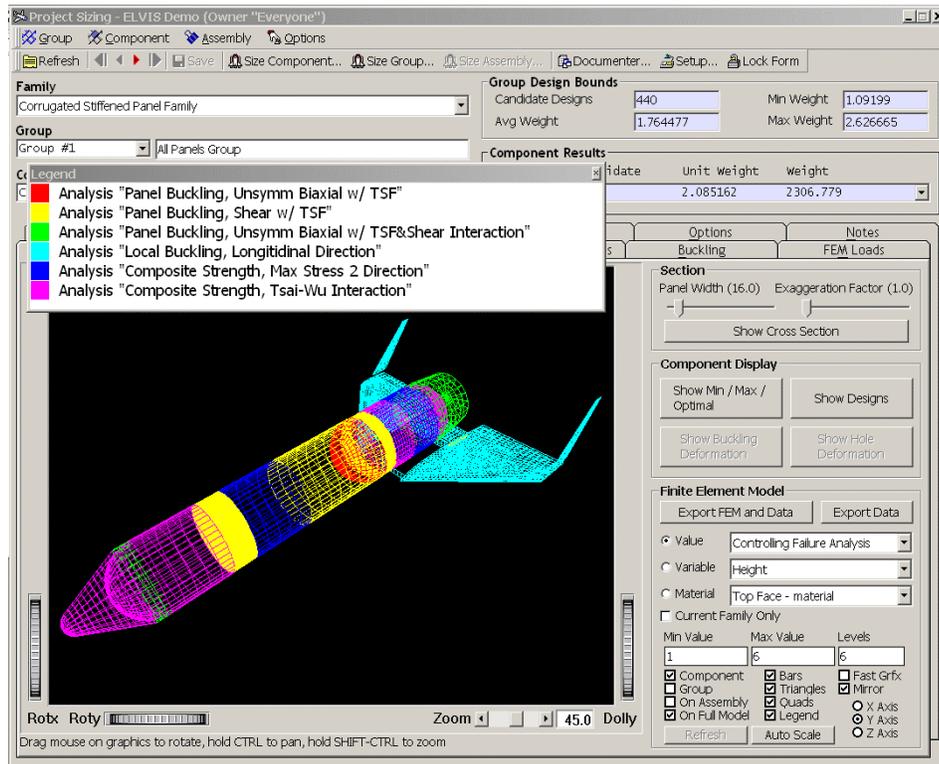


Figure 13: Structural sizing results from the HyperSizer ModelCenter component

## ***Other Potential Applications of the Object Model***

The HyperSizer Object Model opens up many opportunities for integration of HyperSizer into other processes or to solve other problems. Some examples include:

- HyperSizer could become a core component in a structural design system such as the Common Structures Workstation at Boeing or the Virtual Prototyping software at Lockheed Martin by calling HyperSizer directly through the Object Model.
- Wing tip displacement optimization – HyperSizer currently has displacement failure criteria on a component by component basis where each component is sized independently of its neighbors. Using the Object Model, displacements and/or curvatures for components along the span of a wing could be chained to obtain a global displacement criterion.
- Flywheel optimization – The loads on flywheel structural components could be dynamically updated and passed to HyperSizer to account for centripetal force.
- Global optimization response surface generation – The ability of the Object Model to be called many times for local sizing optimization without user-interaction enables the generation of response surfaces for perturbations on design variables. This would allow HyperSizer results to be used in tools like MSC/NASTRAN® Solution 200 or Genesis® from Vanderplaats Research and Development for global optimization.
- Elaborate equations defined in MathCAD or MatLab for defining internal loads based on trajectory and geometric shape parameters.

### **References**

- [1] Niu, M. *Airframe Structural Design*, Conmilit Press Ltd., 1988, p. 378.