SANDIA REPORT

SAND2006-xxxx
Unclassified Unlimited Release
Printed March 2006

The Tool Set for Building Claro - A Component Loading Architecture

Karl G. Merkley, Elemental Technologies, Inc.

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone:	(865) 576-8401
Facsimile:	(865) 576-5728
E-Mail:	reports@adonis.osti.gov
Online ordering:	http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone:	(800) 553-6847
Facsimile:	(703) 605-6900
E-Mail:	orders@ntis.fedworld.gov
Online ordering:	http://www.ntis.gov/ordering.htm

SAND2006-xxxx
Unclassified Unlimited Release
Printed March 2006

The Tool Set for Building Claro - A Component Loading Architecture

Karl G. Merkley
Elemental Technologies, Inc.
17 Merchant Street
American Fork, UT 84003
karl@elemtech.com

Abstract

Claro, A Component Loading Architecture, was designed with two purposes. The first was to provide a graphical user interface for an existing command line tool, Cubit. Second, it was designed to provide a versatile user interface for other tools in the design through analysis (D2A) tool set. While many tools may not fit into the Claro architecture, the tools that were used to build Claro are valid in very broad range of applications. This report discusses the tools (specifically, Qt, VTK, and python) used to build Claro and shows how their utility and versatilty make them applicable to the larger set of D2A tools. As multiple projects use the same set of tools it makes it more probable that different groups can leverage the work of previous developers.

This page intentionally left blank.

Contents

Figures

This page intentionally left blank.

The Tool Set for Building Claro - A Component Loading Architecture

Introduction

Cubit is a finite-element mesh generation tool developed by Sandia National Laboratories [6] that has served as a research platform for important mesh generation algorithms [13][15][17] and for production meshing. Cubit was originally developed as a command-line driven program; however, in 2003 it was determined that to increase the utility and usability of Cubit it should be augmented by a modern graphical user interface (GUI). The requirements for this project extended beyond just providing a GUI for Cubit, they also set forth guidelines for interaction between other design-to-analysis (D2A) applications. The goal was to create a framework for multiple D2A applications. Cubit was designed to be one component that could work within this framework which was designated as Claro. However, Claro is more than just the software package. As the acronym suggests, it is an architecture and a set of tools that extends past the particular software implementation. This paper will focus on three of the major requirements that drove the Claro development process, 1) multi-platform GUI, 2) visualization, and 3) scripting, and discuss the tools that were selected to meet the requirements. While other D2A projects may not chose to use the specific Claro implementation, we believe that the set of tools that have been selected have wide applicability. By selecting a common set of tools, it is anticipated that projects can leverage work within the broad range of D2A applications.

Multi-Platform GUI

Cubit originated over a decade ago and there were strong requirements that it run on multiple operating systems and hardware platforms. At that time there was not a good solution for a graphical user-interface (GUI) that would provide a single code base on multiple operating systems. Caterpillar Inc. developed a Microsoft Windows implementation of a GUI for Cubit and this version was made available to Cubit users within Sandia; however, the Caterpillar GUI for Cubit never gained widespread acceptance. One reason for this lack of acceptance was the lack of support for multiple operating systems and hardware platforms. There are times when a finite element problem requires more resources than are available on a desktop system. Users already face enough challenges when moving to a different

operating system. They do not want the user interfaces of their software packages to change also.

Within the Claro framework we have resolved this issue by using Qt from Trolltech AS [1]. Qt provides a multi-platform user interface solution that allows for a single code base across all platforms. It uses native GUI widgets (buttons, menus, etc.) so the application looks like a native application to the user. It provides good tools for graphically creating the user interface and excellent tools for laying out the user interface so that that sizing is maintained across platforms. Qt also implements a unique method for tying GUI events to functions called the signal-slot interface. The signal-slot interface provides a very intuitive and flexible method for connecting the events to functions. It does not have the disadvantages of the Motif callback method which requires a fixed parameter interface and difficulty in passing custom data and it avoids the problems of ever increasing entries in a switch statement in the GUI event loop.

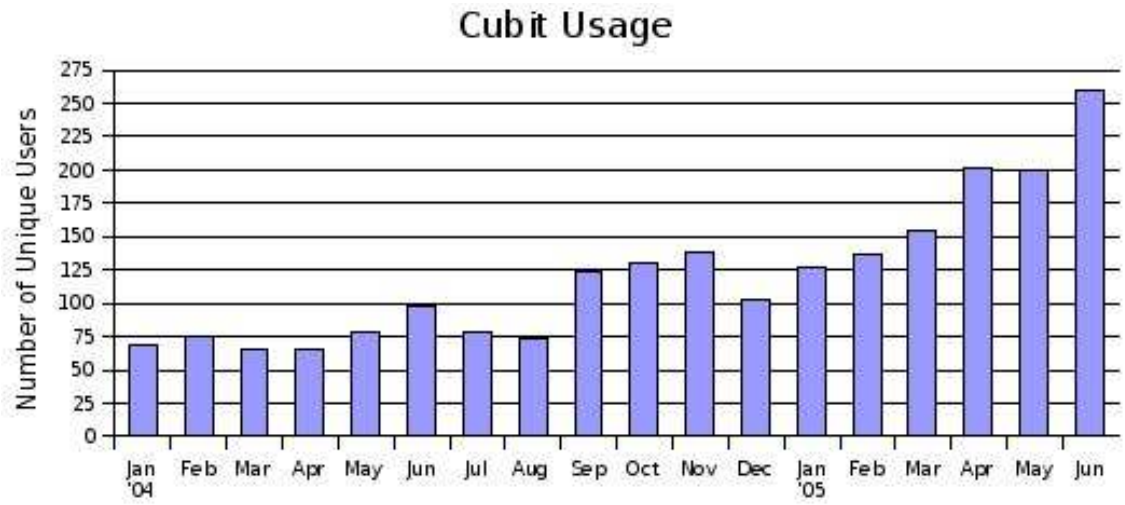

Figure 1. Cubit usage data after the release of the multi-platform GUI.

Figure 1 shows the effects the cross platform GUI has had on Cubit usage. The figure shows the number of unique users that accessed Cubit at least twice in the given month. If a user started up Cubit just to look at the GUI and then never used it again, that user is not counted in the statistics. The first release of the cross-platform GUI was Cubit 9.0 in September of 2004. This release supported Windows, Linux, and Sun. There is an immediate jump in usage at that point. The next release was Cubit 10.0 in May 2005 although beta users had access to it earlier. This release added HP-UX and SGI IRIX to the supported platforms. Once again, there is a corresponding jump in usage.

We have been very successful with several Qt based projects [8]. We have found

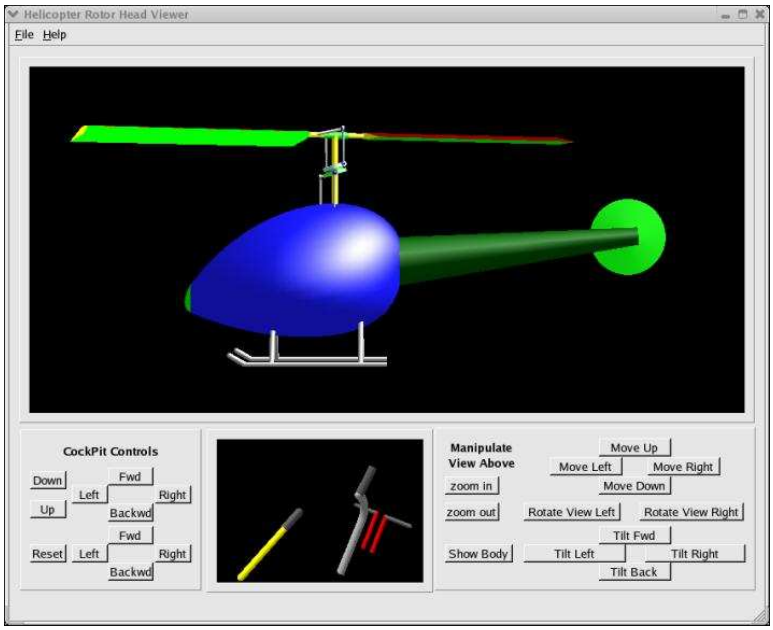

Figure 2. Student project developed with the Qt multi-platform user interface library.

the API easy to use and the implementation to be very solid. Anecdotally, I have found that students in my course at Brigham Young University, Computer-Aided Engineering Software Development, have been much more successful in completing difficult projects since switching the class to Qt. It appears that the GUI programming does not create an obstacle to accomplishing the final project. In fact, it appears that the well organized Qt API assists students in designing better code.

Figure 2 shows a sample student project completed with Qt. It represents a mechanism analysis of a helicopter rotor. The project was completed by a mechanical engineering student with a strong kinematics background but no computer science or programming background prior to completing the course. The project represents the final for the class and was completed in less than four weeks. This is one example of the utility and usability of the Qt interface.

Qt has a dual license. It is released both as GPL and as a commercial license. The GPL license is intended for individuals who are working on developing open source code without remuneration. The commercial license is for groups or individuals that are being paid to develop code that may be sold commercially. We have chosen to license Qt commercially to avoid any potential licensing concerns. We also believe that it is wise to support an entity that provides commercial quality code and support so that the code base will continue to improve.

Visualization

Almost all steps of the D2A process are enhanced by graphical representations of simulation models. Whether the visualization involves constructing a CAD model, meshing, material properties plots, or post-processing, each phase is augmented by some type of visual tool. Computer graphics and scientific visualization are inherent to the Claro architecture.

To support the visualization requirements of Claro we chose to use the Visualization Toolkit (VTK) by Kitware [11]. VTK is a high level graphical library that allows users to piece together a visualization pipeline that includes data sources, filters (or data modifiers), and rendering tools. Complex graphical images can be created with relatively few lines of code. It is important to differentiate between the graphics pipeline and the visualization pipeline. The graphics pipeline in modern systems is typically implemented in the Graphics Processing Unit (GPU) on the video card. It handles the transformation matrices necessary to convert a 3D point in world coordinate space to a pixel on the screen. The visualization pipeline in VTK is at a much higher level than the graphics pipeline. It manipulates large pieces of data and provides data filters to modify and display the data.

For example, figure 3 shows the entire graphics pipeline for creating and displaying a cylinder in VTK.

```
void draw_cylinder()
{
  source = vtkCylinderSource::New();   // Geometry

  mapper = vtkPolyDataMapper::New();   // Mapper
  mapper->SetInputConnection(source->GetOutputPort());

  actor = vtkActor::New();     // Actor in scene
  actor->SetMapper(mapper);

  vtkRenderer* ren = vtkRenderer::New();  // Renderer

  ren->AddActor(actor);  // Add Actor to renderer
  ren->ResetCamera();    // Reset camera
  ren->GetRenderWindow()->Render();
}
```

Figure 3. Sample VTK code for creating and rendering a cylinder.

First a geometry source is defined. In this case the source is a cylinder primitive. The geometry could come from a number of other different sources such as data read from a file in a number of different formats or created as low level entities, points, lines, etc. Next vtkPolyDataMapper is created which converts polygonal data to graphics primitives. The primitives are added to a vtkActor which is an object that is rendered in the final scene. Finally, we add the actor to the rendering scene, reset the camera, and render the image. VTK provides a concise, clear method of producing complex graphical scenes. It removes much of the tedium of managing the entire OpenGL pipeline.

One piece that was missing from the VTK library was support to link Qt and VTK. This support has been developed in the form of the QVKTWidget with funding from the Cubit project as part of the Claro implementation. The QVTKWidget has been placed as open source code into the VTK repository and is freely available. The QVTKWidget provides support for adding graphical widgets interactively via the Qt Designer as well as being added via code. The QVTKWidget was designed to compartmentalize Qt and VTK and allow each library to handle those tasks that they were designed to handle. The QVTKWidget is being used in a number of projects at Sandia [10] and around the world [7][9].

VTK is open source with a BSD license. It can be freely used anywhere. There is excellent documentation on the web [5] and an active mailing list that provides support for VTK.

Scripting

Another requirement that drove the Claro architecture was the ability to create a scriptable interface. Cubit already contains the Algebraic Pre-Processor (APREPRO) [12] and it has proven to be very useful in creating parametric models. However, the success of APREPRO has led to many enhancement requests for features like looping and tests that make it verge on being a true scripting language. We determined that it would be useful to use a widely accepted scripting language rather than to continue to invent and maintain our own language.

There were a number of requirements that were placed on the choice of a scripting language. It had to be widely accepted. It had to provide a complete set of programming features. It had to be embeddable in a C++ executable. It had to interface with Qt so that users could customize the interface as well as create parameterized models.

After doing extensive research we chose to use the Python [16] scripting language as the basis for scripting in Claro. Python is widely used within the scientific and

engineering communities. Commercial codes such as Abaqus [4] have adopted Python as the scripting language of choice. Lawrence Livermore National Laboratories has had a great deal of experience with Python in scientific computing [14] and found it be a very useful tool. We chose to follow a path that was well defined.

One issue that was required was a method for exposing methods in a C++ library to the scripting language. We chose a tool called the Simplified Wrapper and Interface Generator, SWIG, to provide an interface [2][3]. We create a very simple interface file that includes the interface header file that is to be wrapped. We purposely defined the interface to be very simple. It contains strings, ints, doubles, and vectors or arrays of those quantities. SWIG will generate this interface automatically without any additional coding.

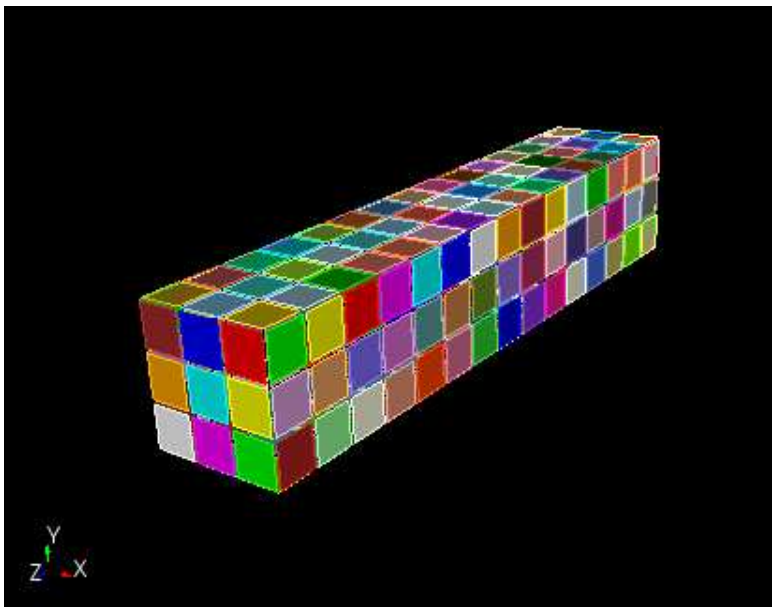

Figure 4. Example of automatic boundary condition definition for a gas chomatograph analysis.

We have found the scripting interface to very useful. It provides a method of delivering code to specific users needing functionality without having to wait for an entire release cycle. It also provides a method of delivering code to users that is of interest only to that user and is not of benefit to the general user base. Cubit has been continually developed for over a decade. In that period of time, a number of users have made requests for functionality to support specific research topics. Once the functionality has been added to the code base, it is very difficult to remove. As a result this code must be maintained over the lifetime of the code at significant expense. By delivering user specific functionality as a script, we remove this code from the Cubit code base and place any maintenance burden on the user. The user can then determine when the need for the functionality has elapsed. If the functionality proves to be of use to the general user base, it is possible to migrate

the algorithm from the scripting language to the core C++ code base.

Figure 4 demonstrates an example of the scripting capabilities. The structure represents a model of a gas chromatograph. The geometry is very simple. It is no more than a rectangular prism. However, the boundary conditions require marking the opposite faces of each chord of hexahedral elements in the model. Picking the opposite faces by hand is a very tedious process and the user requested help in automating the process. The colored faces in 4 show the boundary sets. Faces on opposite sides of the prism have the same color indicating that faces form a pair. For this example, the number of elements and the length of the tube have been decreased significantly. This is very specific functionality that saves the user hours of work for each analysis but it is not of use to the general user base.

The core of the script that performs this operation is shown in figure 5.

```
for surface_id in surface_list:
 # get all the quads on the surface
  surface_quads = cubit.get_surface_quads(surface_id)

  for quad in surface_quads:
    # get a chord of hexes starting from the given quad
    start_quad_nodes = cubit.get_connectivity("Quad", quad)
    hex_chord = get_hex_chord(start_quad_nodes)

     # find the last hex and quad nodes the chord
    last_hex, opp_quad_nodes = \
         find_opposing_face_nodes(hex_chord, start_quad_nodes)

    # get all the dimension 2 (quad) elements off the hex
    quad_list = cubit.get_sub_elements("hex", last_hex, 2)

    # find the quad associated with nodes on the end of the chord
    opposite_quad = find_face(quad_list, opp_quad_nodes)
    command = "sideset " + str(start_sideset_id) + " face " + \
          str(quad) + " " + str(opposite_quad)
    cubit.cmd(command)  # execute the command in cubit
    start_sideset_id += 1
```

Figure 5. A portion of the code for finding opposite faces in a hexahedral prism .

Since Cubit was originally developed as a command driven system and the original command language must be maintained to provide backwards compatibility, it was decided to retain that command language structure for modifying the model

state. Currently, there is one main modify entry point and that entry point takes a Cubit command string. This allows us to access all the Cubit modify and creation functionality without having to build additional interfaces.

Python and SWIG are both open source codes with a BSD type license. There are no restrictions on the use of these tools.

Conclusion

Claro is both a software implementation and an encapsulation of some architectural requirements for software supporting the design through analysis process. Some of the major requirements that Claro supports are a multi-platform graphical user interface, high-level visualization tools, and scripting. In the Claro development process we have had very good success meeting the requirements by specifying and using the proper set of tools. The tools that we have used for meeting these requirements are respectively: Qt, VTK, and Python. By using the proper set of tools we have been able to leverage the efforts of multiple developers both inside and outside of the Claro project, increase productivity and meet user needs.

References

[1] http://www.trolltech.com, 2001.

[2] D. Beazley. http://www.swig.org, 2005.

[3] T.L. Cottom. Using SWIG to bind C++ to Python. *Computing in Science and Engineering*, 5(2):88–97, 2003.

[4] Abaqus Inc. *Abaqus 6.5 User's Manual*. Abaqus Inc., www.abaqus.com, 2005.

[5] Kitware Inc. Vtk 5.0.0 documentation. Technical report, oct 2005.

[6] Sandia National Laboratories. http://cubit.sandia.gov, 2005.

[7] J. Linxweiler. Ein prototyp zur immersiven betrachtung und interaktiven manipulation rŁumlicher objekte. Master's thesis, Technische UniversitŁt Braunschweig, July 2005.

[8] K. Merkley, R. Meyers, C. Stimpson, and C. Ernst. Verde - VERification of Discrete Elements. In B.K. Soni and J.F. Thompson, editors, *8th International Conference on Numerical Grid Generation in Computational Field Simulations*. International Society of Grid Generation, 2002.

[9] O.Pironneau. http://www.freefem.org/ff3d, 2005.

[10] D. H. Rogers and C. J. Garasi. Prism: A multi-view visualization tool for multiphysics simulation. In *3rd International Conference on Coordinated and Multiple Views in Exploratory Visualization*. IEEE, July 2005.

[11] William J. Schroder, Kenneth M. Martin, and Lisa S. Avila. *VTK User's Guide*, 2003.

[12] G. D. Sjaardema. APREPRO: An algebraic preprocessor for parameterizing finite element analyses. Technical Report SAND92-2291, Sandia National Laboratories, Albuqurque, NM, 1992.

[13] M. B. Stephenson T. D. Blacker. Paving: A new approach to automated quadrilateral mesh generation. *International Journal for Numerical Methods in Engineering*, 32:811–847, 1991.

[14] P.F. Dubois T.-Y.B.Yang, G. Furnish. Steering object-oriented scientific computations. In *TOOLS 23. Technology of Object-Oriented Languages and Systems*, pages 112–119, July 1997.

[15] S.A. Mitchell T.J. Tautges, T. Blacker. The whisker weaving algorithm: A connectivity-based method for constructing all-hexahedral finite element meshes. *International Journal for Numerical Methods in Engineering*, 39:3327–3349, 1996.

[16] Guido van Rossum. http://www.python.org, 2005.

[17] D.R. White and T.J. Tautges. Automatic scheme selection for toolkit hex meshing. *International Journal for Numerical Methods in Engineering*, 49(1):127–144, September 2000.

DISTRIBUTION:

1 Karl G. Merkley
Elemental Technologies, Inc.
17 Merchant Street
American Fork, UT 84003

1 MS 0376
Ted Blacker, 1422

1 MS 0376
Steve Owen, 1422

1 MS 0822
Dave White, 9227

1 MS 0822
Dave White, 9227

2 MS 9018
Central Technical Files, 8945-1

2 MS 0899
Technical Library, 4536