

WATT: A Compiler for Automated Visualization Service Generation

Evan F. Bollig¹, Martin D. Lyness², Gordon
Erlebacher¹ and David A. Yuen²

GCE '07

11 November 2007

Reno, NV

¹ School of Computational Science, Florida State University

² Minnesota Supercomputing Institute, University of Minnesota



Background

- Large consortiums investigate specialized topics with similar problems:
 - Task automation (workflows, load balancers, compilers, etc.)
 - Public access to utilities (repositories, portals, etc.)
 - Distributed computation
- In VLab (<http://vlab.msi.umn.edu>) everything should be a service
 - Compute services
 - Task management services
 - Visualization services

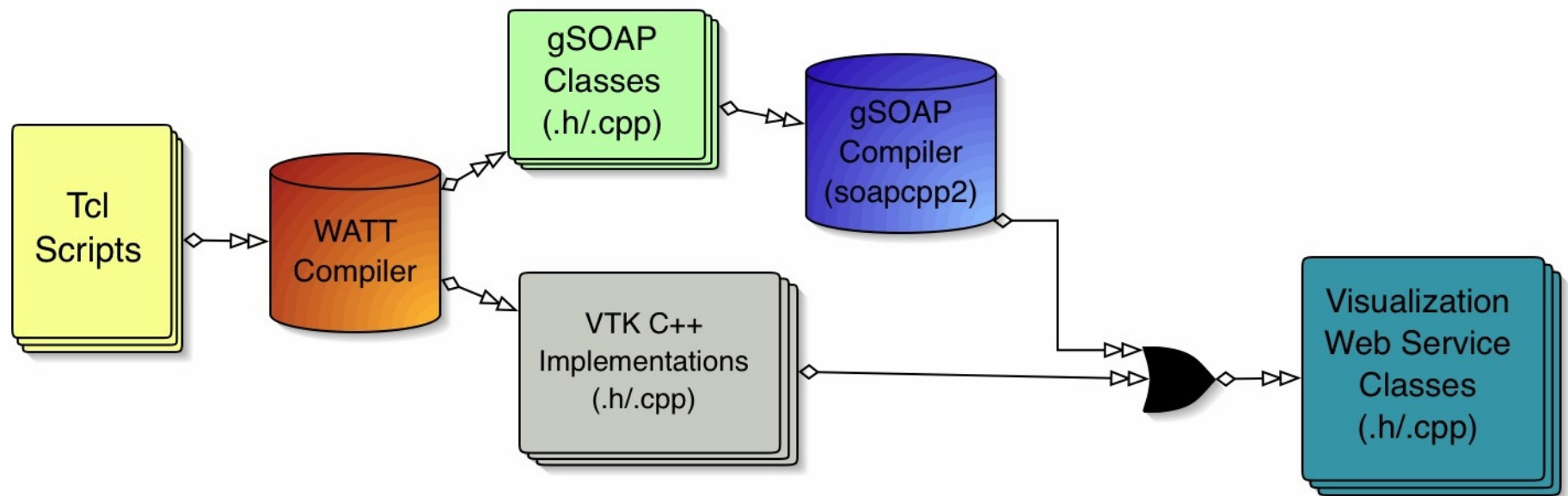
Can we automate service creation?

- Process to write a visualization service:
 1. Write VTK (Tcl) script to render example data
 2. Convert Tcl to C++ (by hand)
 3. Add service details (SOAP, WSDL, security, etc.)
 4. Compile and deploy(Repeat)
- 2 and 4 stay the same; 3 rarely changes

The Web Automation and Translation Toolkit (WATT)

- “Compiler” (> Translator)
 - Tcl to C++
 - Requires type inference
 - Adds gSOAP to C++
 - gSOAP provides SOAP handlers, WSDL, stub and skeleton
 - WATT connects gSOAP to core code
 - Code Generation
- Developers write Tcl script, WATT does the rest. (Almost...)

WATT Execution



Defining the Stub

- Standard set of published methods
 - Allow clients calling standard methods to work with any service
 - `renderBase64Binary` → returns image as byte array
 - `jumpToView` → move camera to view all rendered objects along specified vector
 - and more...
- Tcl “proc” commands
 - Expand functionality of service
 - Can be associated with UI toggles, value modifiers, etc.

Determining Types

- Register system provides type mapping hints
- “proc” types determined by last statement and parameter usage
- Two phase type inference:
 - Generate abstract syntax tree and match known types with registers
 - Walk tree again assigning types based on other usages

Input Limitations

- Limited Tcl syntax
 - No variables in strings (i.e. “hello \$name”)
 - No variables in names (i.e. set my\$name)
 - No loops or conditionals
- VTK classes (and member functions) must be registered or compilation fails
- Use WATT for a head start on the final C++ program.
 - Unsupported content can be added by hand after rest is compiled.
 - Charge Density service followed this approach.

Templates

- C++ code fragments
- Variables
 - \$\$VARIABLE_NAME\$\$
 - replaced with buffered content from WATT
- Contain pre-written standard methods
- Structure and content of template can change; default template structure shown here

```
{Includes}
{WATT_CLASS Definition
  [Global Declarations]
  [Constructor]
  [Member Functions]}
{Main Method}
{Web Method Wrappers}
```

NOTE: each line can contain one or more variables

Simple Example (VisQuad.tcl)

```
package require vtk
{...}
vtkQuadric quadric
  quadric SetCoefficients .5 1 .2 0 .1 0 0 .2 0 0
{...}
vtkContourFilter contours
  contours GenerateValues 5 0.0 1.2
{...}
```

```
#WATT_EOF
```

← WATT stops parsing here

```
iren AddObserver UserEvent {wm deiconify .vtkInteract}
```

```
iren Initialize
```

```
wm withdraw .
```

**NOTE: original is 33 lines (65 w/ comments);
WATT output is 300+ lines C++**

Simple Example (Cont'd)

// Includes:

```
#include "vtkQuadric.h"  
#include "vtkContourFilter.h"  
    (...)
```

// WATT_CLASS Definition:

```
class MyWattClass {
```

// Global Declarations:

```
    vtkQuadric *quadric;  
    vtkContourFilter *contours;  
    (...)
```

// Constructor:

```
    MyWattClass () {  
        (...)  
        quadric = vtkQuadric::New();  
        quadric->SetCoefficients((float) .5,  
            (float) 1, (float) .2, (float) 0, (float)  
            .1, (float) 0, (float) 0, (float) .2,  
            (float) 0, (float) 0);  
        (...)  
        contours = vtkContourFilter::New();
```

```
        contours->GenerateValues((int) 5, (float) 0.0,  
            (float) 1.2);  
        (...)
```

```
    } // end Constructor
```

// Member Functions:

```
    vtkUnsignedCharArray* render () {...}  
    (...)
```

```
    } *wattObject;
```

// Main Method:

```
int main( int argc, char *argv[] )  
{  
    (...)
```

```
    wattObject = new MyWattClass();  
    runSerialSoap( port, "MyService.wsdl" );  
}
```

// Web Method Wrappers:

```
int ns__render (SOAP* soap,  
    ns__renderResponse* result) {...}  
    (...)
```

WATT Live

- Perl CGI scripts to test custom Tcl with WATT
- Generated service available as download.
- Port range for testing services (and clients)
- VisQuad service image shown here

Testing Client + Service Connection

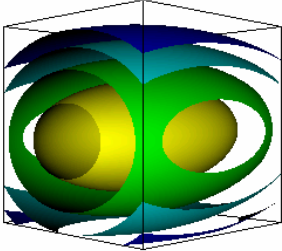
Client Log

Running Client:
r - force render
j - set camera position vector (viewing vector from origin) and render
q - quit
Enter vector along which the camera will move to view data (x,y,z): vector set.
read: 20256 tuples
Wrote tuples to file.

Execution Details

```
source ./watt-exec/wattVars.sh; cd /tmp/watt/test56; echo "j 1 0 1 q" | ./Client.bin -p 1989
```

Test Image (View from 1 0 1)

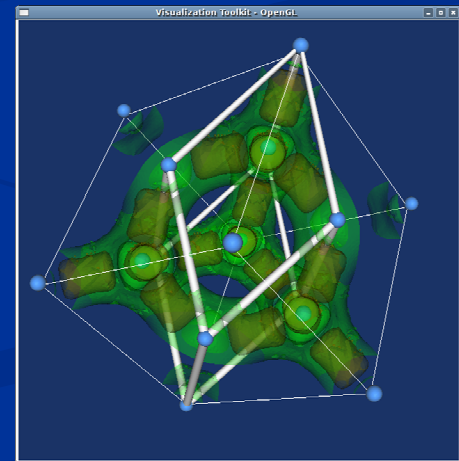
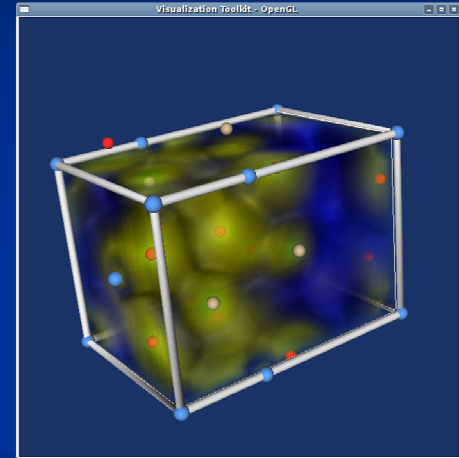


Change Camera Position Vector

XYZ

Charge Density Service

- 400+ line Tcl script for VLab
 - View Plane-Wave self consistent field (PWscf) charge densities as volume, isosurface and orthoslice representations.
 - Reads Gaussian Cube format
- Procedures to toggle representations and update values
- Original test for WATT
 - Available as example for WATT Live



Charge Density Portlet

The screenshot shows a web browser window displaying the 'Charge Density Portlet' within the 'Gridsphere portal framework'. The browser address bar shows the URL: `http://llli.msi.umn.edu:8080/gridsphere/gridsphere?cid=cdvcPortlet&gs_action=`. The page title is 'Charge Density Visualization Client'. The interface is divided into several sections:

- Project Status:** Shows details for project 'new-single', including 'type of input' (Single), 'status' (input completed, submitted, execution complete), and 'description' (created on: Sat, Aug 4 2008).
- Project Overview:** Lists project entries under categories: 'submitted' (new-single, test-single-8-24), 'ready to submit' (new-eos), 'incomplete input' (test-ht, new-eos-2), and 'no input' (junit-selector-080810630).
- ChargeDensity from Project new-single:** Contains a 'Load Data Set' button and a URL input field with the value `http://www.scs.fsu.edu/~bollig/si.cube.dat`.
- Visualization:** A 3D visualization of a charge density surface, rendered in green and yellow, with a dashed wireframe bounding box. The surface shows a complex, multi-lobed structure.
- Quick Views:** A dropdown menu set to '(1,1,1)'. Below it, 'Current View (X, Y, Z):' has input fields for X, Y, and Z, all set to '1', and an 'Update' button.
- Options:** A list of toggleable options: Atoms (On), Volume (On), Contours (On), CutPlane (On), PrimitiveCell (On), BoundingBox (On), and Off.
- Cut Plane Depth:** An input field set to '0.50'.
- Footer:** A link to 'Kill JavaScript'.

The status bar at the bottom of the browser window shows: `t?action=jumpToView¶ms=1,1,0&endpoint=http://llli.msi.umn.edu:1988' time:`

- Gridsphere portlet provides controls; Axis servlet in back-end calls service.
- Manually written
- Dashed lines distinguish UI components.
- Orthoslice depth control depends on manual changes to back-end service.

Automatic Client Generation

- `#WATT_GUI {...}` directives within “proc” content describe UI components
 - SETVALUE
 - PRESET
 - TOGGLE
 - RENDERED_IMAGE
- Components common to most UI packages
 - HTML, Qt, Swing, etc.
- Ruby-on-Rails utility (work in progress) uses WATT generated file to create Ajaxified web interface

The screenshot shows a web browser window with the URL `http://watt.gorerle.com/code/gui`. The page title is "Gui for Watt Client". The interface is divided into two main sections: a control panel on the left and a 3D visualization on the right.

Control Panel (Left):

- Visualization Options:** A list of checkboxes with labels: `show_volume` (unchecked), `show_cut_plane` (unchecked), `show_bounding_box` (checked), `show_contours` (checked), `show_atoms` (checked), and `show_primitive_cell` (checked).
- Add Contour:** A section with a label "Add Contour" and input fields for "Min: .0010556", "Max: .0867451", and "Nb: 8".
- Set Contour:** A section with a label "Set Contour" and input fields for "index: fill-in" and "value: fill_in".
- Set Sample Resolution:** A section with a label "Set Sample Resolution" and input fields for "resx: 128", "resy: 128", and "resz: 128".
- Resolution Presets:** A section with a label "Resolution Presets" and a dropdown menu showing "128x128x128".
- Set View:** A section with a label "Set View" and input fields for "x: 1", "y: 1", and "z: 1".
- View Presets:** A section with a label "View Presets" and a dropdown menu showing "xyz* (111)".

3D Visualization (Right):

A 3D visualization of a crystal structure, showing a complex, multi-faceted shape with a green and blue color gradient. The structure is enclosed within a white wireframe bounding box. The visualization is set against a dark blue background.

Kill-A-WATT (KWATT)

- Work in progress to replace WATT
- Stand-alone web service
 - Service spawns new services
- Services (C++) control Tcl interpreter
 - Tcl was written to extend C++
 - Interpreter provides full access to Tcl types and commands without limitation
 - No registers or type hints
 - Use any package with Tcl bindings
 - Automate compute service generation
 - Procedure content can be registered in the interpreter at any time
 - Patch/update service methods without downtime

Recap

- WATT
 - Automates service development.
 - Many limitations still present. KWATT to address these problems.
- WATT Live
 - Compiler as a service
- Client Interface Generation
 - Create user interfaces based on stub description
 - Possible to generate local and web based interfaces from one description

Acknowledgements

- Thanks to the Virtual Laboratory for Earth and Planetary Materials (VLab) for their support
<http://vlab.msi.umn.edu>
- This work is supported by NSF through the ITR grant NSF-0426867