Modeling Frameworks, Workflows and Community Modeling: Where are we now and where do we go from here?

Scott D. Peckham
Chief Software Architect for CSDMS
University of Colorado, Boulder
April 23, 2013

csdms.colorado.edu

What are the Key Problems ?

Heterogeneity: Lots of resources (data, models, web services, etc.), but all different. How do you accommodate differences between resources? (names, data types, units, grids, timesteps, formats…)

Discovery: How do you find a resource of interest? (e.g. ontologies)

Linking/Connecting: Models have input and output variables, data has only "output". How do you connect the users to the providers? How much data is involved? What is the *data transfer rate*?

Semantic issues: Perhaps the "*last frontier*" of interoperability. We need to boldly go there and develop *domain-independent* solutions.

Performance, speed & efficiency: The 80/20 Problem: 80% grunt work, 20% doing science. Automation and seamless integration can turn this around. Need to design around data transfer rates.

Some Key Concepts and Terms

Model: *State variables* for some system of interest are discretized in space (on a *computational grid*) and new values are computed from previous values by *marching forward in time* according to a set of rules (e.g. laws of physics).

Model Component: A model that has been specially prepared for plug-and-play reusability. (i.e. wrapped with a *standard interface* & compiled for *language interoperability*).

Interface: A standardized *set of functions*, which a caller uses to interact with a resource such as a model, database or service. *Heterogeneous resources* wrapped with a *standard interface* then operate in a "familiar" way.

Modeling Framework: A software container in which pre-compiled model components are instantiated, configured and *dynamically linked* to rapidly *create a customized model*. (With a "birds-eye view" of all components.)

Workflow: Software that allows a *chain of steps* to be saved, modified and re-executed. Each step uses an application to create an *intermediate product* that is passed along to the next step, *often as a file*. (e.g. Kepler, Pegasus)

Taming Heterogeneity with Interfaces

Before:

Each resource is unique.
Own ways of doing things.
Respond differently.
Can become unstable.
Difficult to control.

After:

Uniform outward appearance.
Respond to same commands.
Interchangeable units.
Have a chain of command.
Work as a team.

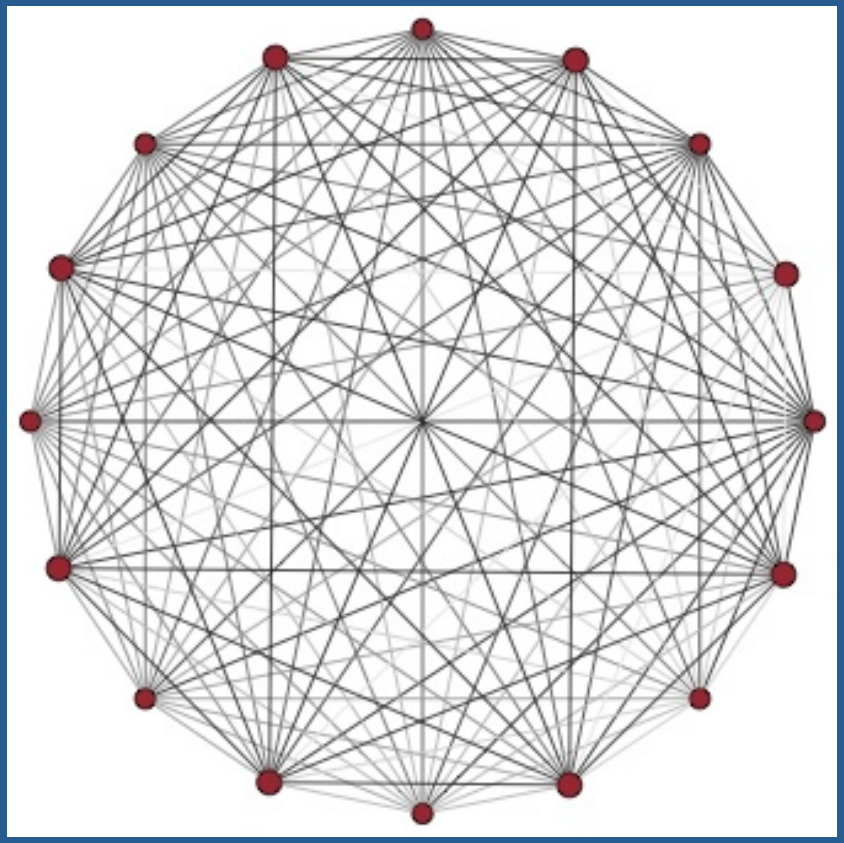
Local and Global Standards

Building bridges between Silos:

Semantics, Formats, Etc.

Different domains (e.g. disciplines) are like silos. They should be free to use their own internal or "**local standards**", but need to agree on some common "**global standards**" when they want to start sharing resources.

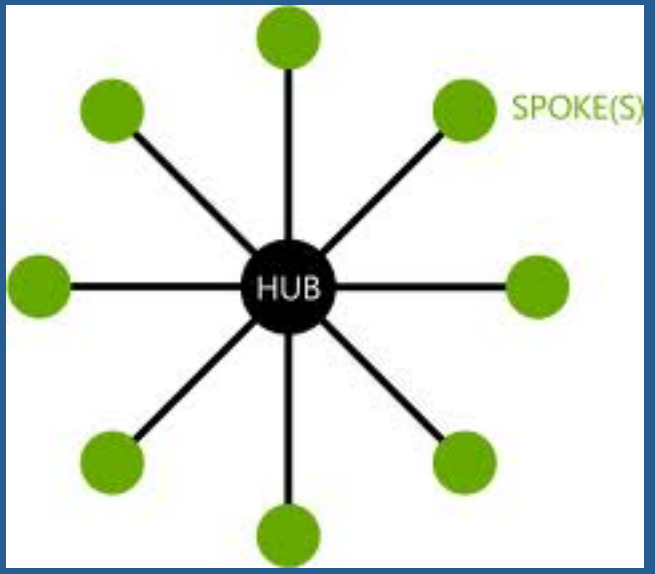
Standards Reduce Complexity



If we reconcile differences between the resources in a pairwise manner, the amount of work, etc. grows fast:

Cost(N) = N (N-1) / 2 ~ N^2.

vs.



Introduce a new, generic or standard representation (the "hub"), then map resources to and from it. The amount of work, maintenance, etc. drops to:

Cost(N) = N.

Typical Data Transfer Rates

Reading data from RAM: (DDR3 SDRAM, sequential) (Writing is about 40 times slower)	4000 MB/sec - 17,000 MB/sec
Reading data from Solid State Drive (SSD): (Write speeds not much slower.)	200 - 600 MB/sec
Reading data from a Hard Drive (HDD): (7200 rpm, seek + read): (SATA = 3 Gbps max= 375 MB/sec, not realized)	50 MB/sec
Transfer data over a *wired* network: (1 Gbps + protocol overhead)	100 MB/sec
Transfer data over a *wireless* network:	5 to 30 MB/sec

These largely determine the appropriate solution or tool for a given problem.

Modeling Frameworks:

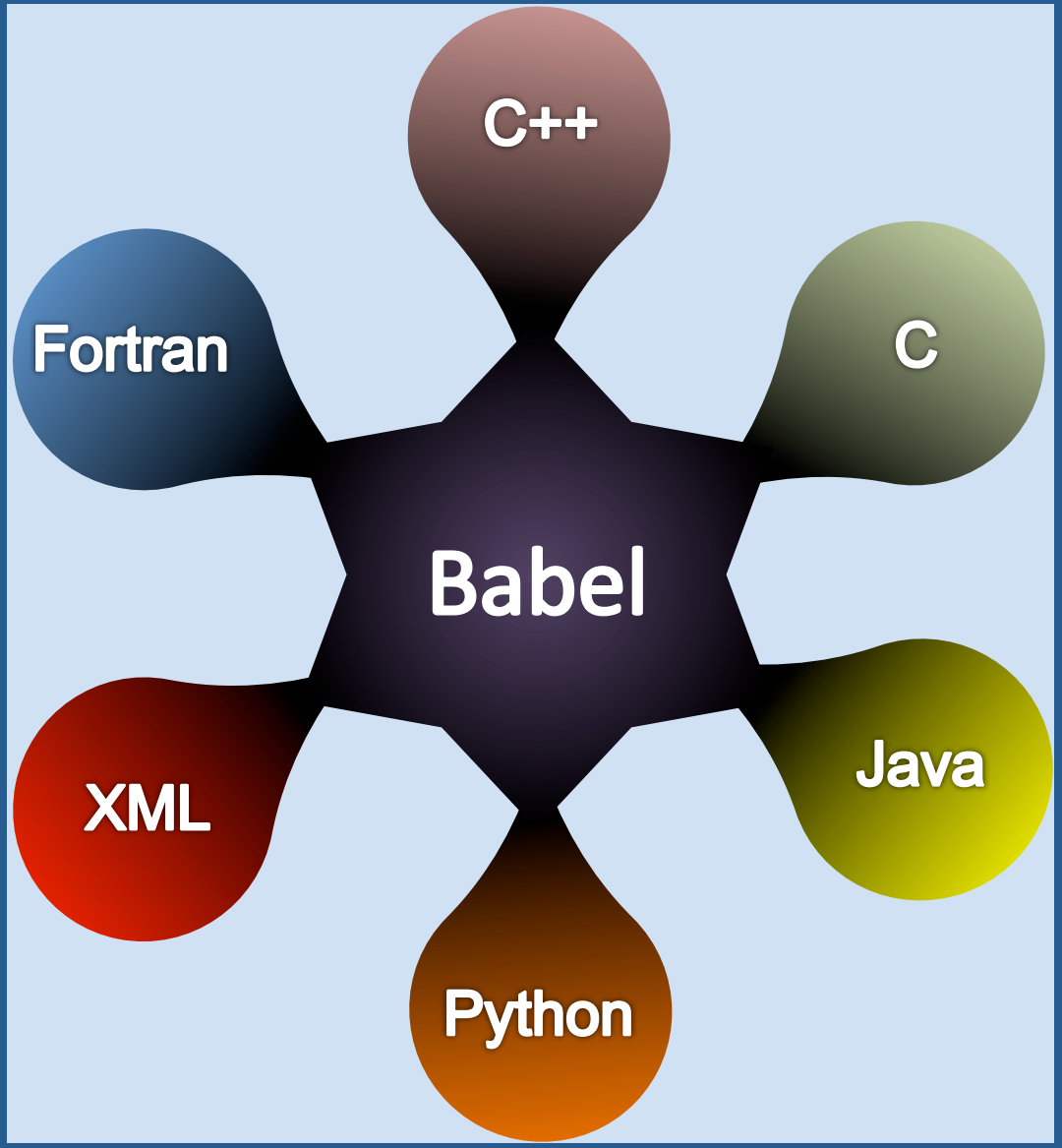
Build a new model from a collection of reusable, pre-compiled, plug-and-play components.

This new model could then become a node in a workflow.

Linking Component-based Models: How Can Two Models Differ?

- Programming language
  (C, C++, Fortran, Java, Python, etc.)
  Solution: Babel and Bocca (CCA toolchain)
- Computational grid
  (triangles, rectangles, Voronoi, etc.)
  Solution: ESMF regridder (parallel, spatial interpol.)
- Timestepping scheme
  (fixed, adaptive, local)
  Solution: Temporal interpolation tool
- Variable names
  Need some means of “semantic mediation”
  Solution: CSDMS Standard Names
- Variable units
  Solution: UDUNITS (Unidata)

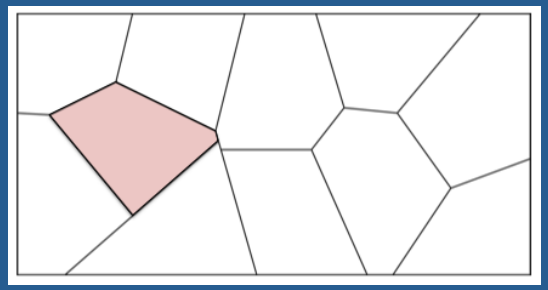
Language Interoperability with Babel

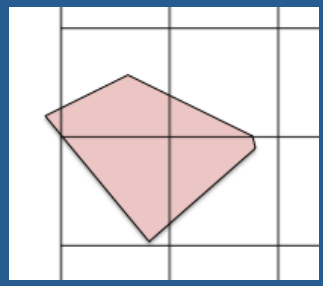


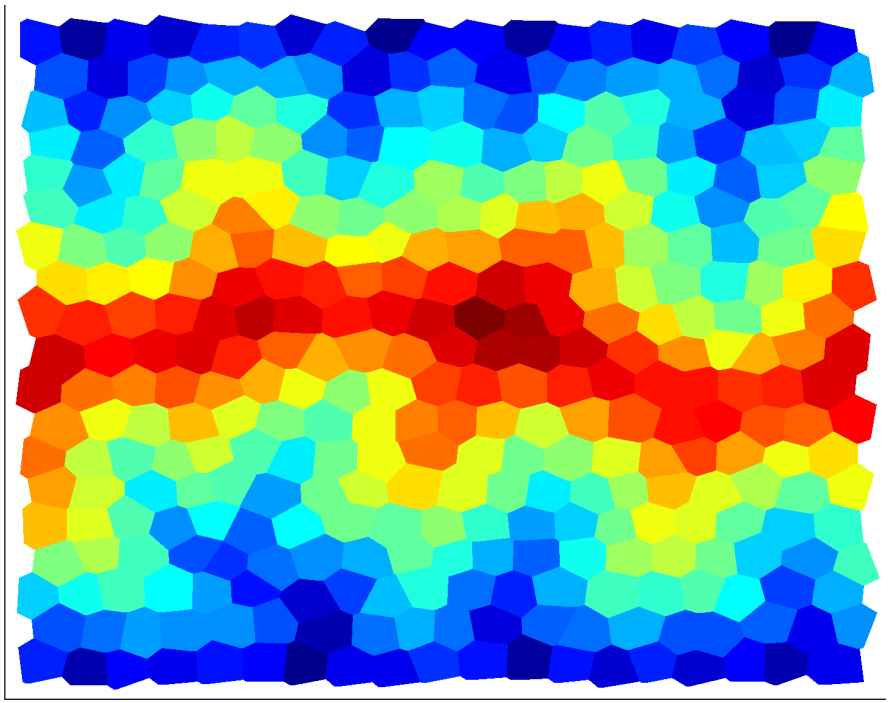
Language interoperability is a powerful feature of the CCA framework. Components written in different languages can be rapidly linked in HPC applications with hardly any performance cost. This allows us to "shop" for open-source solutions (e.g. libraries), gives us access to both procedural and object-oriented strategies (legacy and modern code), and allows us to add graphics & GUIs at will.

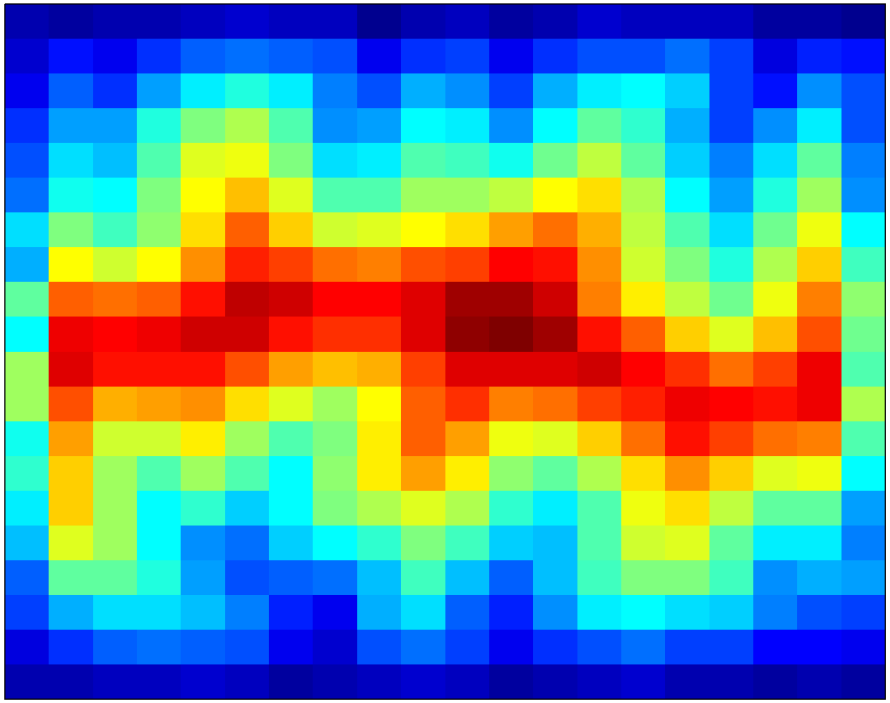
CSDMS Regridding Tools

- ESMF Regrid (multi-proc., Fortran)
- OpenMI Regrid (single-proc., Java)









The Basic Model Interface (BMI) Standard

BMI requires a developer to make some relatively simple, *noninvasive*, and *framework-independent* changes to his/her model source code, mostly by adding some new functions. Functions can be grouped into:

Model Control Functions (initialize, update, finalize)
Model Information Functions (e.g. time-stepping scheme)
Variable Information Functions (e.g. dimensions, data type, units)
Variable Getters and Setters
Grid Information Functions

CSDMS automatically wraps BMI-enabled models to provide them with a CMI interface. CMI makes function calls to the (1) *framework*, (2) *service components (the "reconcilers")*, (3) *the BMI of the wrapped model* and (4) *the CMI of other plug-and-play components*. CMI gives a model many new capabilities, including NetCDF output.

CSDMS Talking to a Model

CSDMS: So, tell me about yourself.

Model:

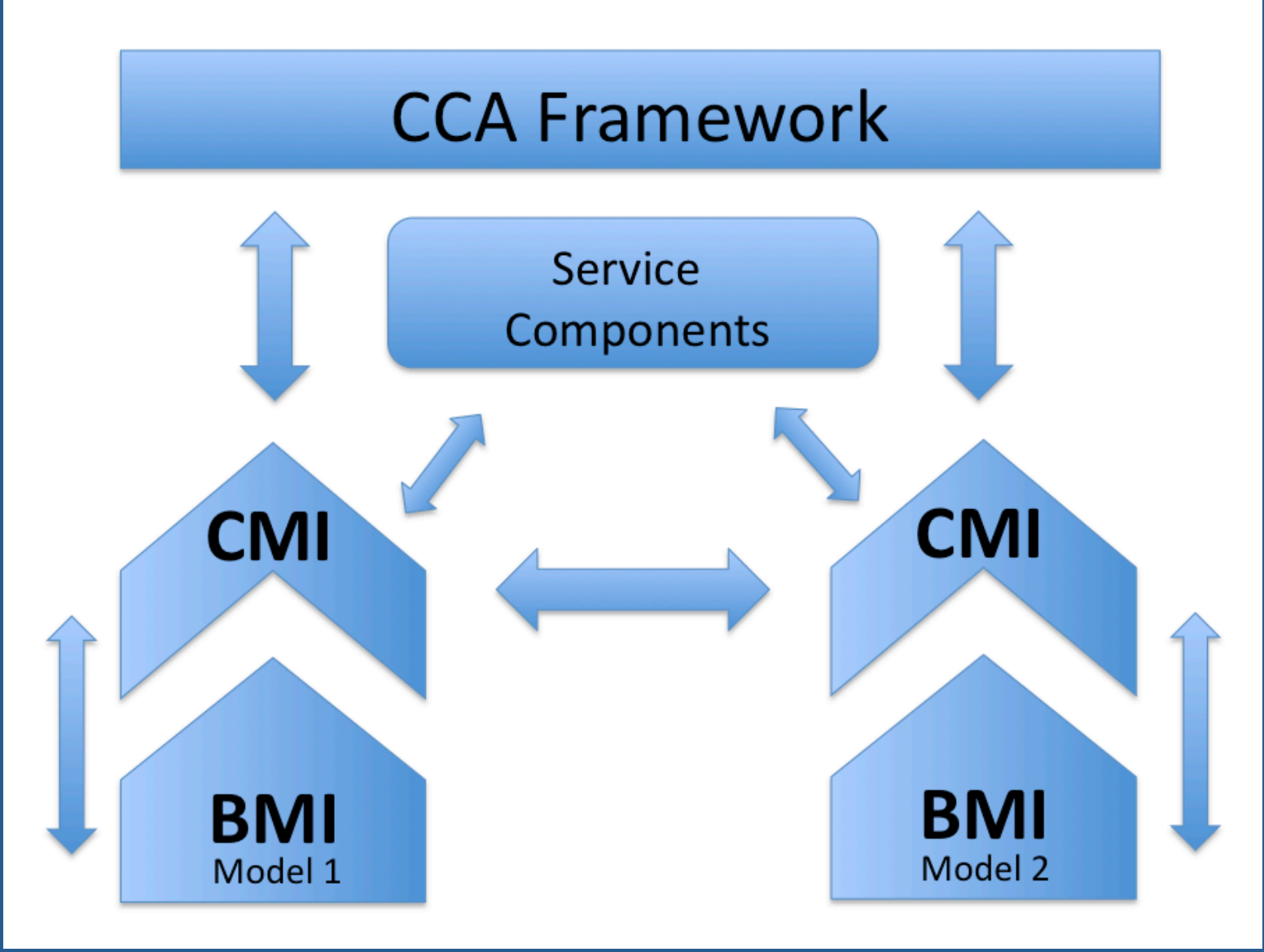
I need input variables A, B and C.
I provide output variables X, Y and Z.
I use adaptive time-stepping.
I use a computational grid of Voronoi polygons.
I like pina coladas and getting caught in the rain.

CSDMS: Those are great attributes. I think we can hook you up.

Model: Am I just an ***object*** to you ??

CSDMS: *Yes*. Your state variables are your *data members*. Your *member functions* consist of the BMI interface functions.

How BMI and CMI Work



The CSDMS Standard Names

Data Models like RDF and EAV use *triples* like:

Subject + Predicate + Object, and

Entity/Object + Attribute + Value (object-oriented)

CSDMS Standard Names use a similar pattern for creating unambiguous and easily understood standard *variable names or "preferred labels"* according to a **set of rules**. These are then used to retrieve/match values (and metadata). The pattern is:

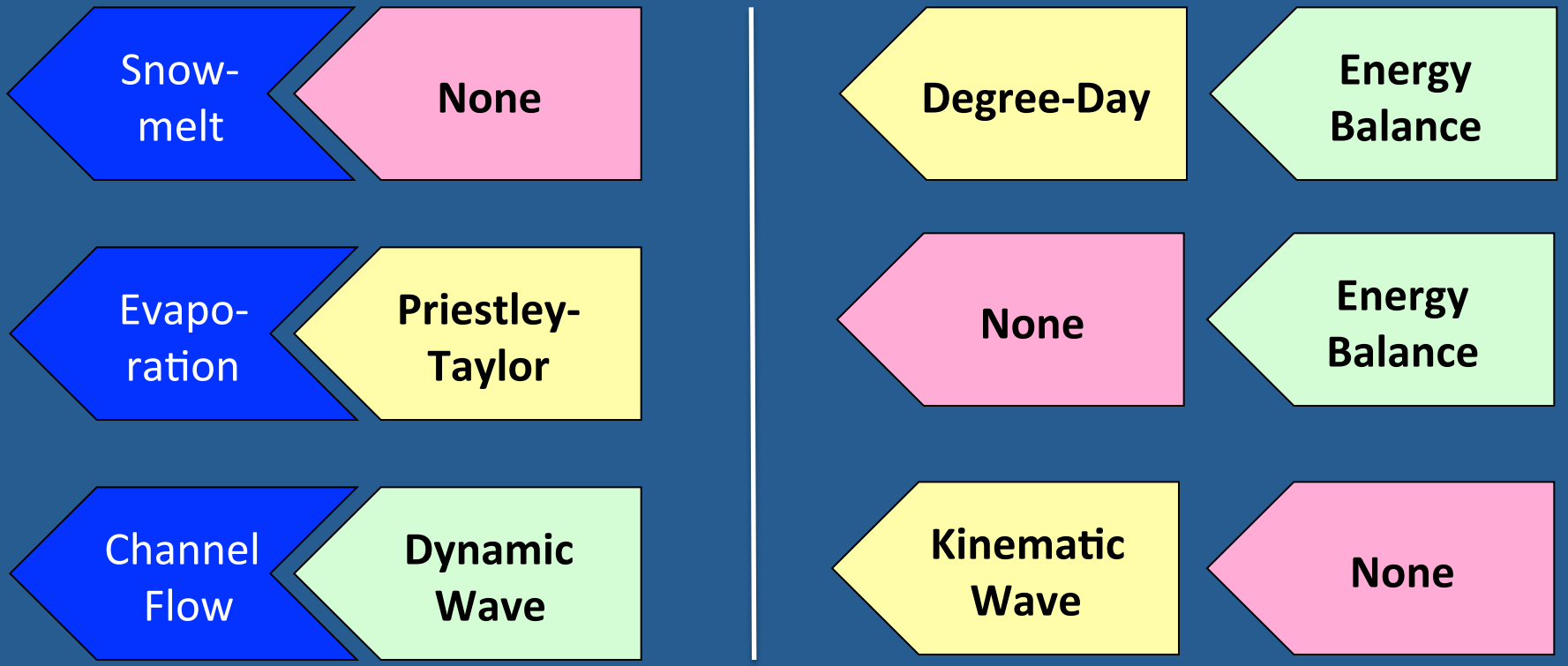
Object name + [Operation name] + Quantity name

Examples:

atmosphere_carbon_dioxide__partial_pressure
atmosphere_water__liquid_equivalent_precipitation_rate
earth_ellipsoid__equatorial_radius
soil__saturated_hydraulic_conductivity

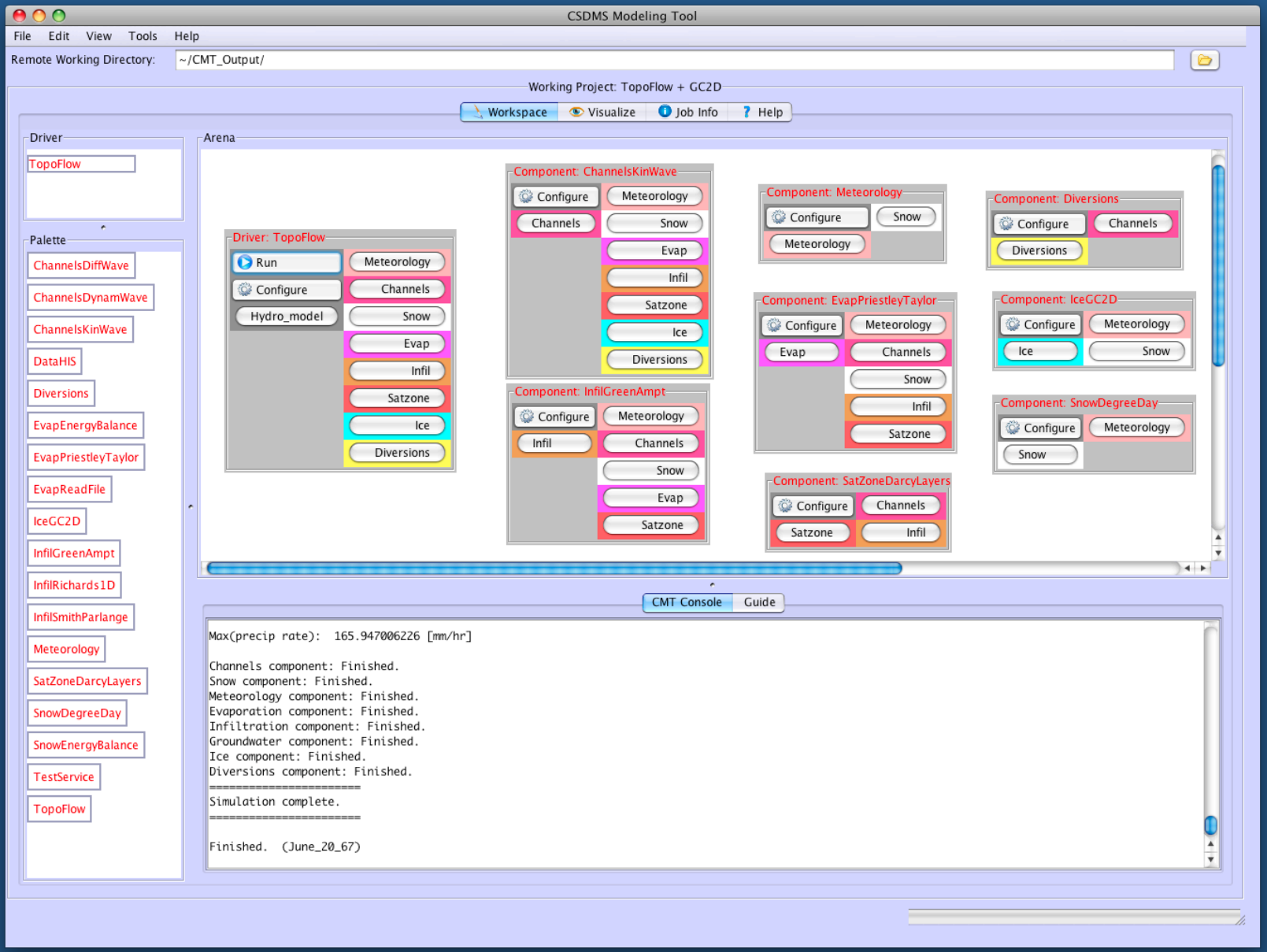
We have also started building a set of standard Attribute and Process Names.

Multiple Methods per Process



For each physical process (blue), there are typically many different ways to parameterize it, ranging from ***not at all*** to ***simple*** (with few inputs) to ***advanced*** (with many inputs).

The CSDMS Modeling Tool (CMT)



For More Information

Peckham, S.D., E.W.H. Hutton and B. Norris (2012)
A component-based approach to integrated modeling in the geosciences: The design of CSDMS,
Computers & Geosciences

Available online at:
http://www.sciencedirect.com/science/article/pii/S0098300412001252

Basic Model Interface (BMI):
http://csdms.colorado.edu/wiki/BMI_Description

CSDMS Standard Names:
http://csdms.colorado.edu/wiki/CSDMS_Standard_Names

Transit of Venus at Sunset: June 5, 2012

Building a Modeling Framework

CSDMS has integrated a variety of powerful, open-source tools to build its modeling framework, such as:

Babel – Language interoperability (C,C++,Java,Python,Fortran)
Bocca – Component preparation and project management
Ccaffeine – Low-level model coupling (parallel environ.)
ESMF Regrid – Multi-processor spatial regridding
OpenMI Regrid – Single-processor spatial regridding
OpenMI – Component interface standard (1.4 and 2.0)
NetCDF – Scientific data format (self-describing, etc.)
VisIt – Visualization of large data sets (multi-proc.)

We greatly extended the original *Ccaffeine GUI* to create our ***CSDMS Modeling Tool*** for interactive model coupling.

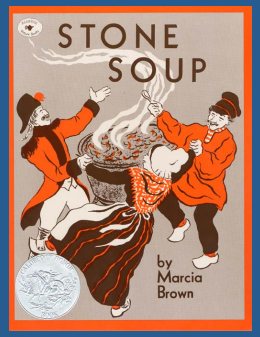
How Can a Model be Converted to a Reusable, Plug-and-Play Component?

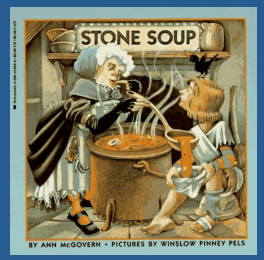
CSDMS has developed two model *interfaces*, one called *Basic Model Interface (BMI)* and one called *Component Model Interface (CMI)* for this purpose.

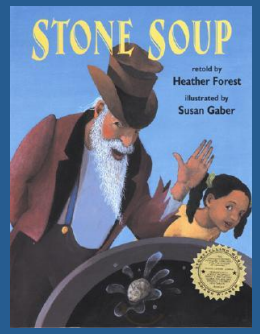
BMI requires a developer to make some relatively simple, *noninvasive*, and *framework-independent* changes to his/her model source code, mostly by adding some new functions. These provide a caller with 3 key things: (1) *fine-grained control (i.e. IUF)*, (2) *variable getters and setters* and (3) *model metadata*. The model can still be used in "stand-alone mode," just as before.

CSDMS automatically wraps BMI-enabled models to provide them with a CMI interface. CMI makes function calls to the (1) *framework*, (2) *service components (the "reconcilers")*, (3) *the BMI of the wrapped model* and (4) *the CMI of other plug-and-play components*. CMI gives a model many new capabilities, including NetCDF output.

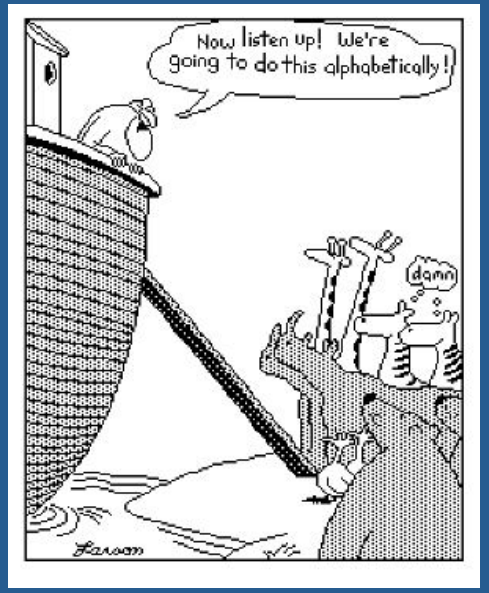
One "Super Model" or Stone Soup?







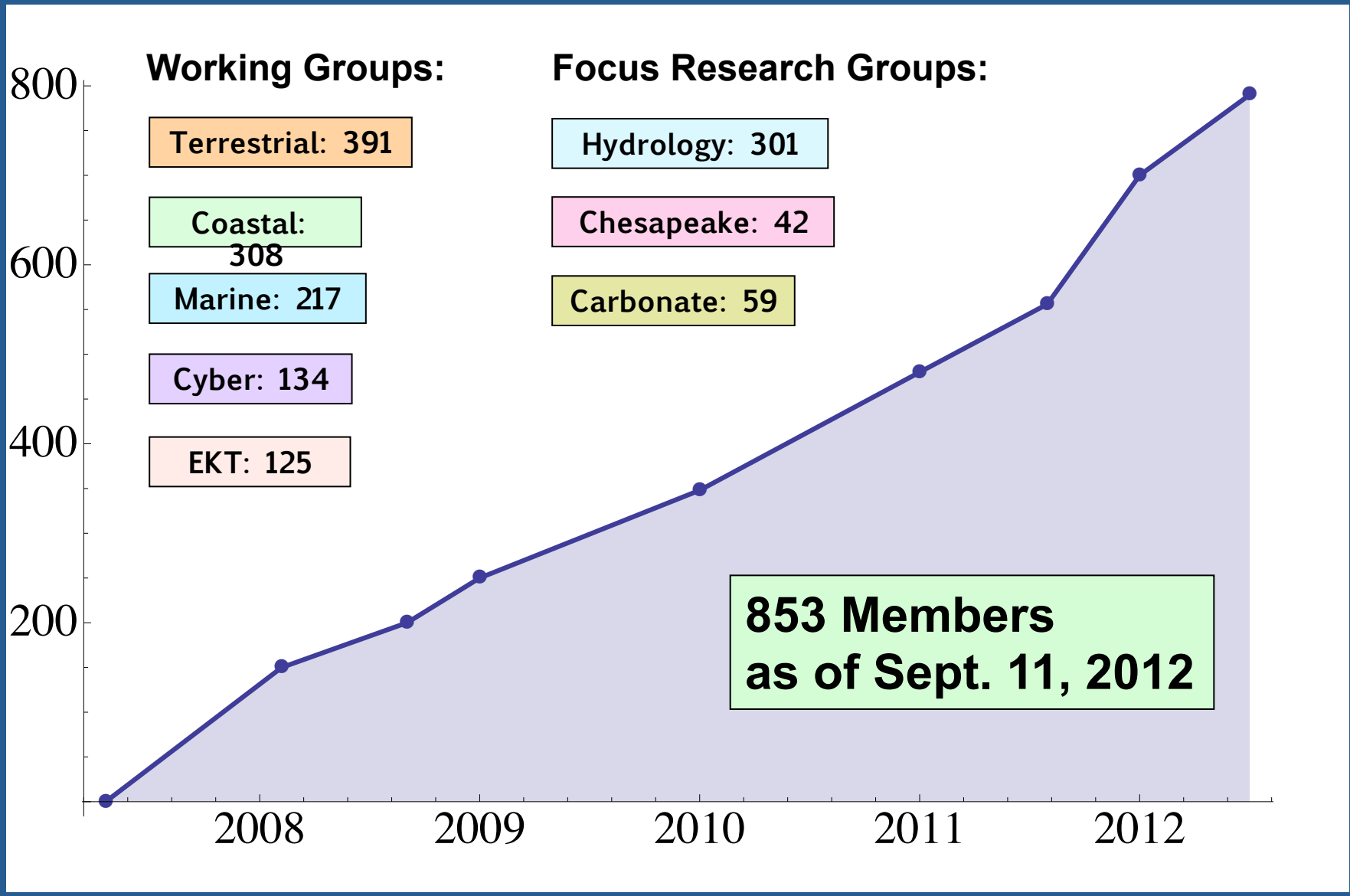
Collecting Earth Surface Process Models



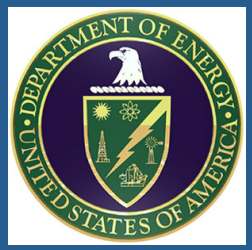
Open source:

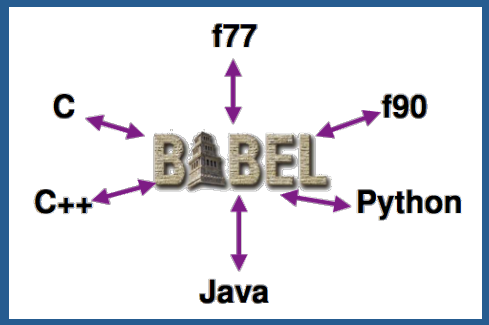
164 Models
51 Tools
55 Components

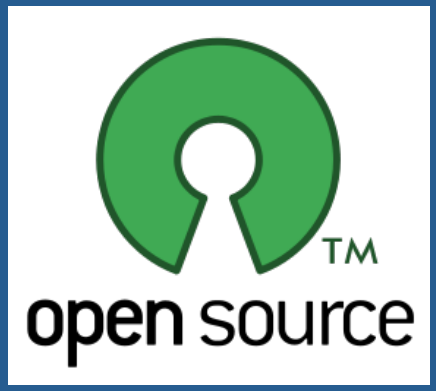
Number of CSDMS Members vs. Time



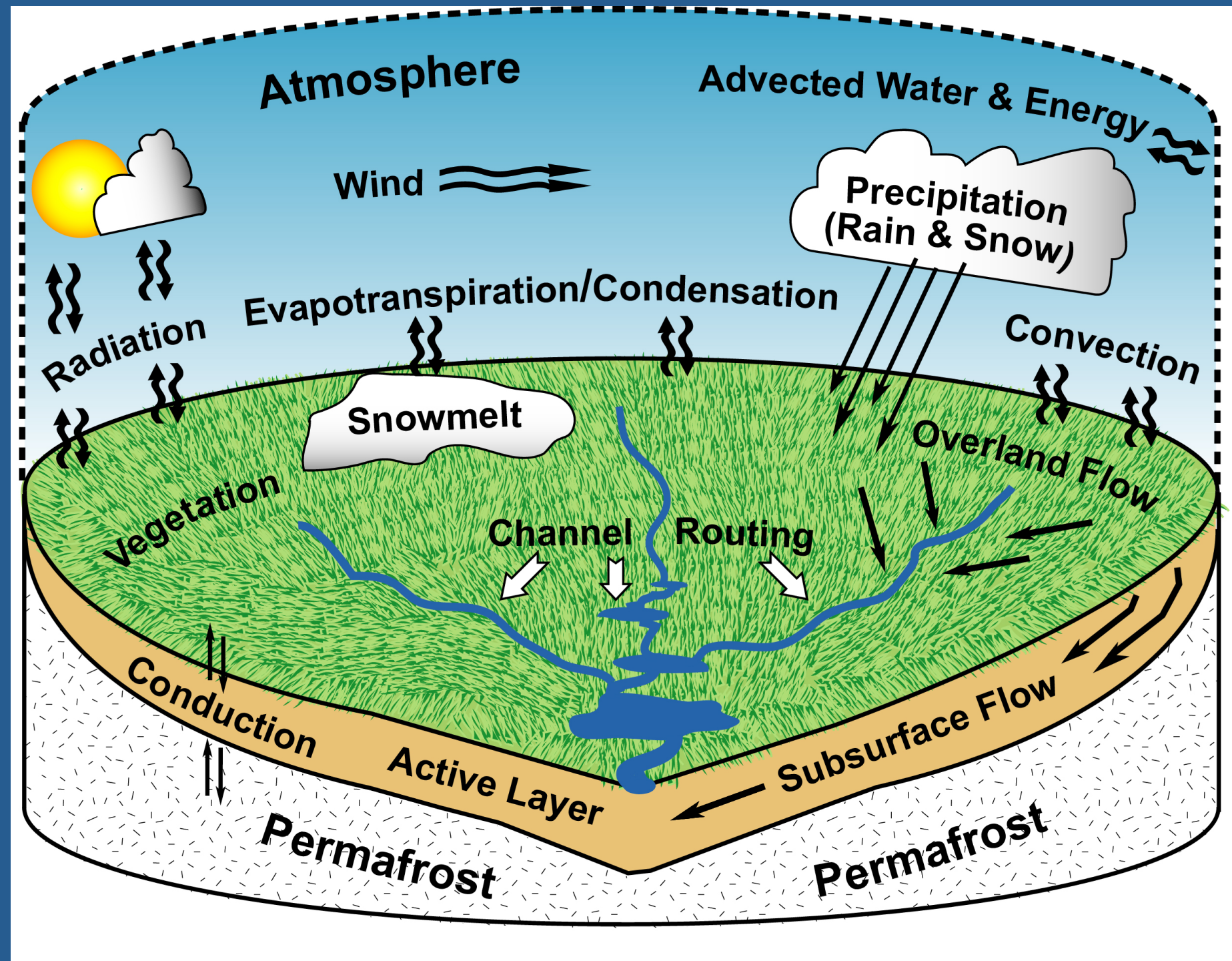
Powerful Tools used by CSDMS



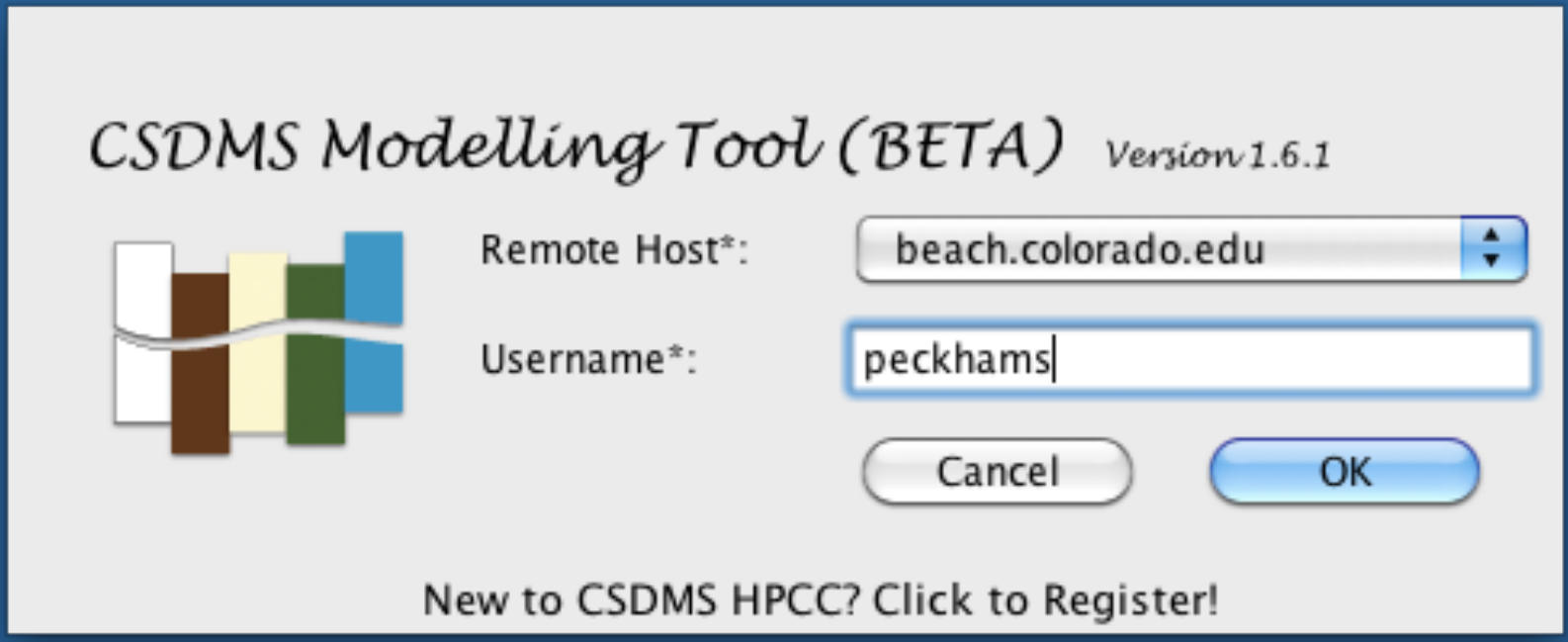




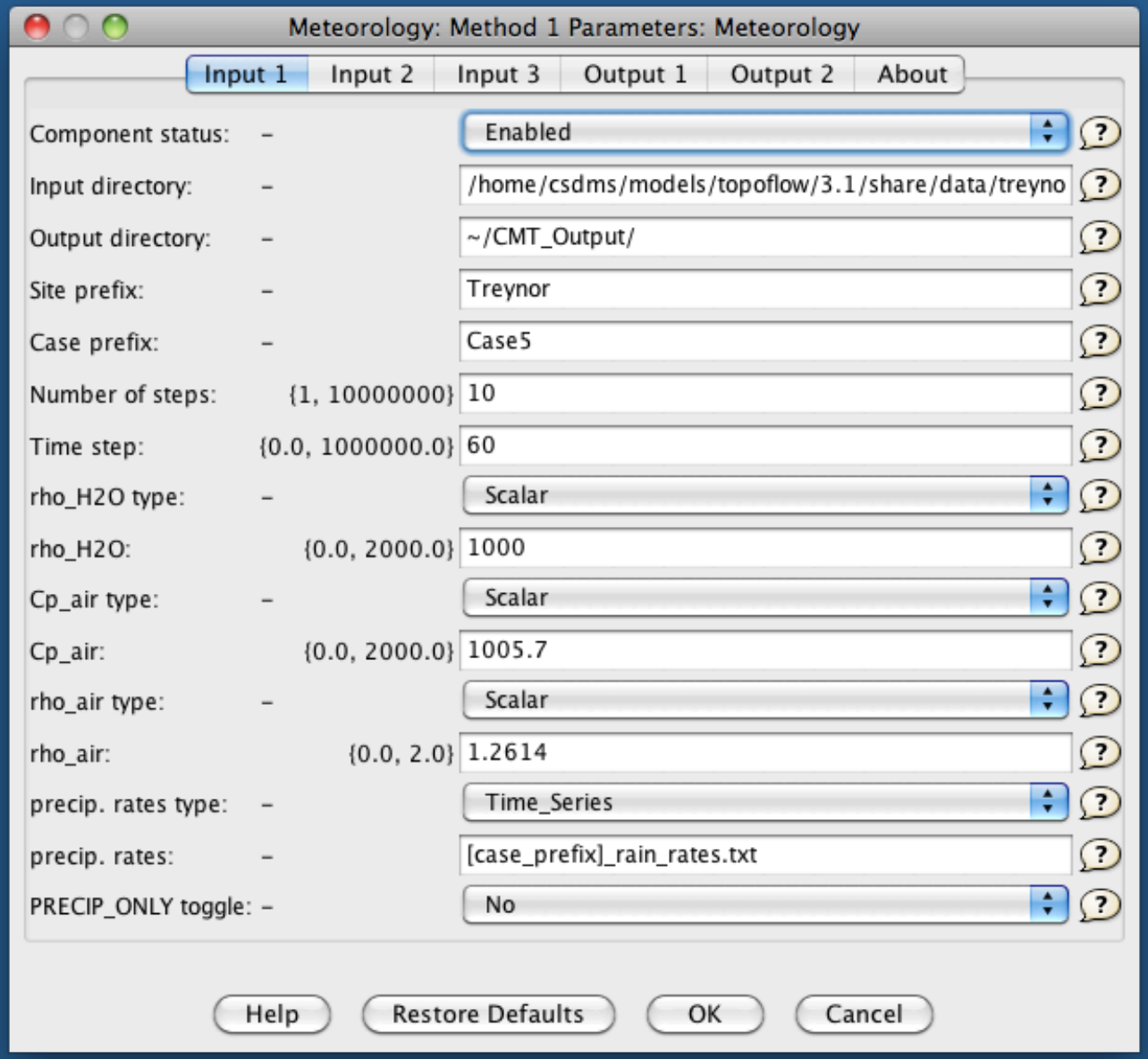
Building and Running a Spatial Hydrologic Model with CMT


Atmosphere
Advected Water & Energy
Wind
Precipitation
(Rain & Snow)
Radiation
Evapotranspiration/Condensation
Convection
Snowmelt
Overland Flow
Vegetation
Channel Routing
Conduction
Active Layer
Subsurface Flow
Permafrost
Permafrost

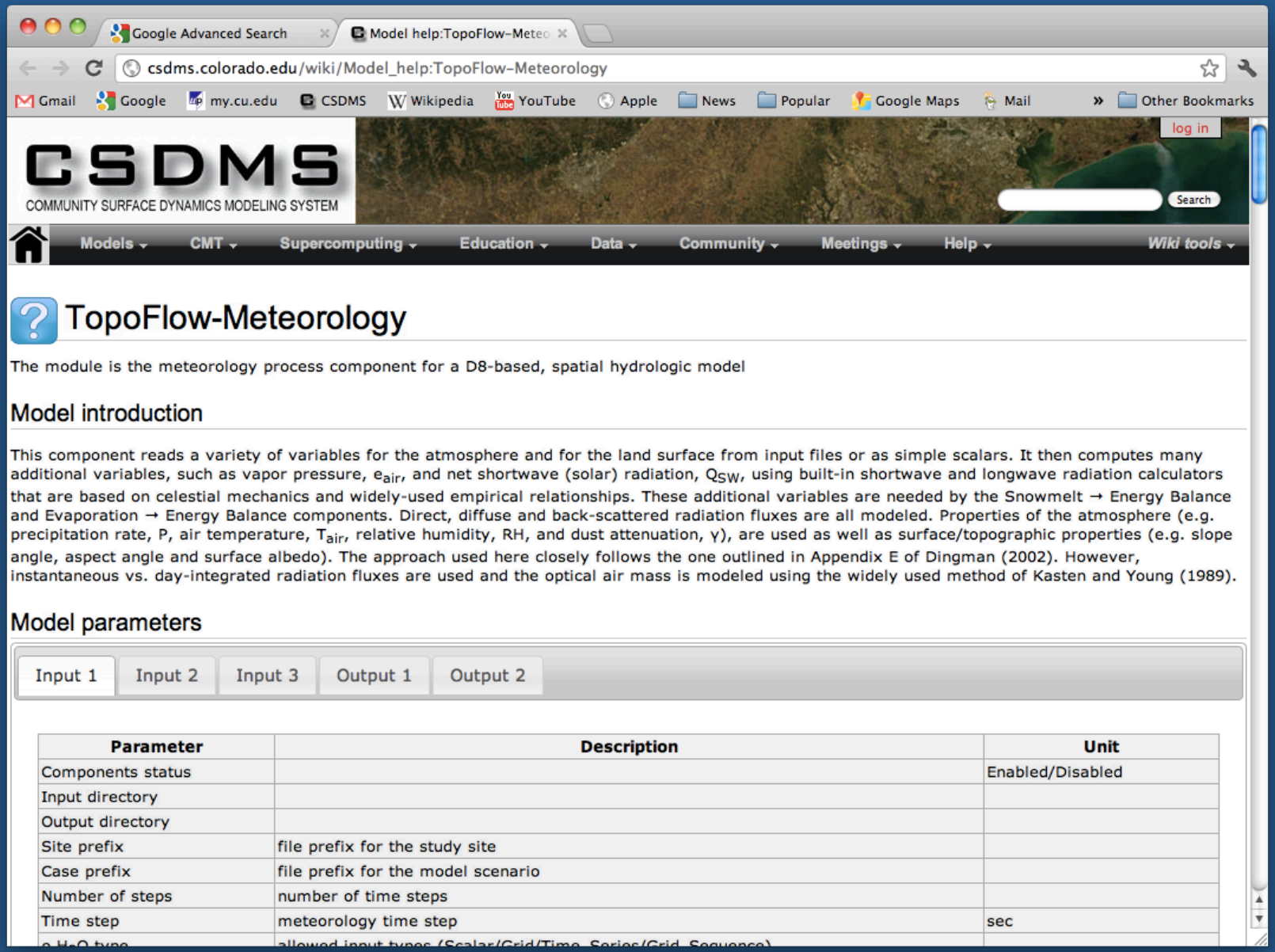
Live Demo of CMT 1.6



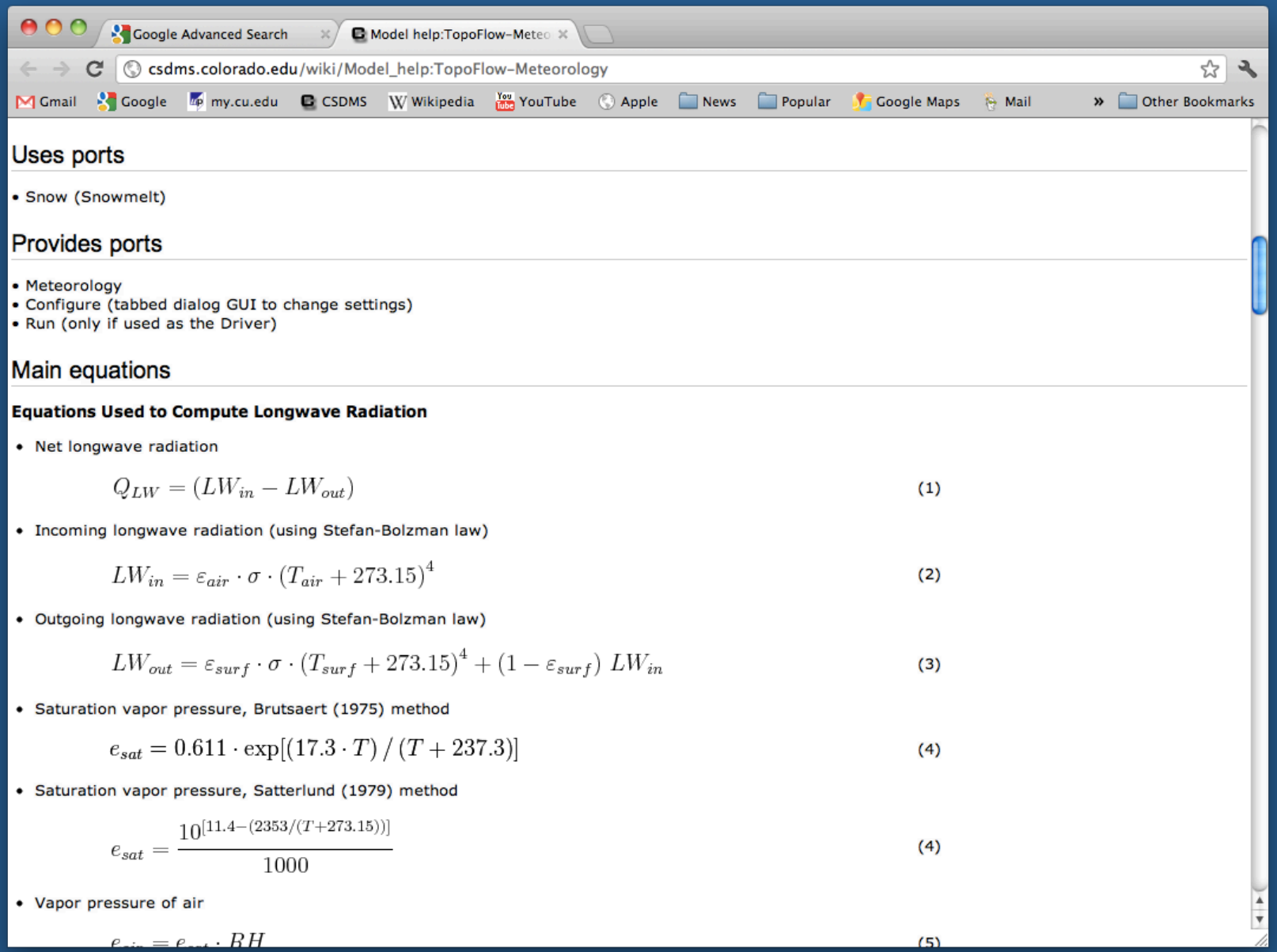
Tabbed Dialog for Meteorology



HTML Help for Meteorology Component



HTML Help for Meteorology Component



Uses ports

- Snow (Snowmelt)

Provides ports

- Meteorology
- Configure (tabbed dialog GUI to change settings)
- Run (only if used as the Driver)

Main equations

Equations Used to Compute Longwave Radiation

- Net longwave radiation

$$Q_{LW} = (LW_{in} - LW_{out}) \quad (1)$$

- Incoming longwave radiation (using Stefan-Bolzman law)

$$LW_{in} = \varepsilon_{air} \cdot \sigma \cdot (T_{air} + 273.15)^4 \quad (2)$$

- Outgoing longwave radiation (using Stefan-Bolzman law)

$$LW_{out} = \varepsilon_{surf} \cdot \sigma \cdot (T_{surf} + 273.15)^4 + (1 - \varepsilon_{surf})\ LW_{in} \quad (3)$$

- Saturation vapor pressure, Brutsaert (1975) method

$$e_{sat} = 0.611 \cdot \exp[(17.3 \cdot T) / (T + 237.3)] \quad (4)$$

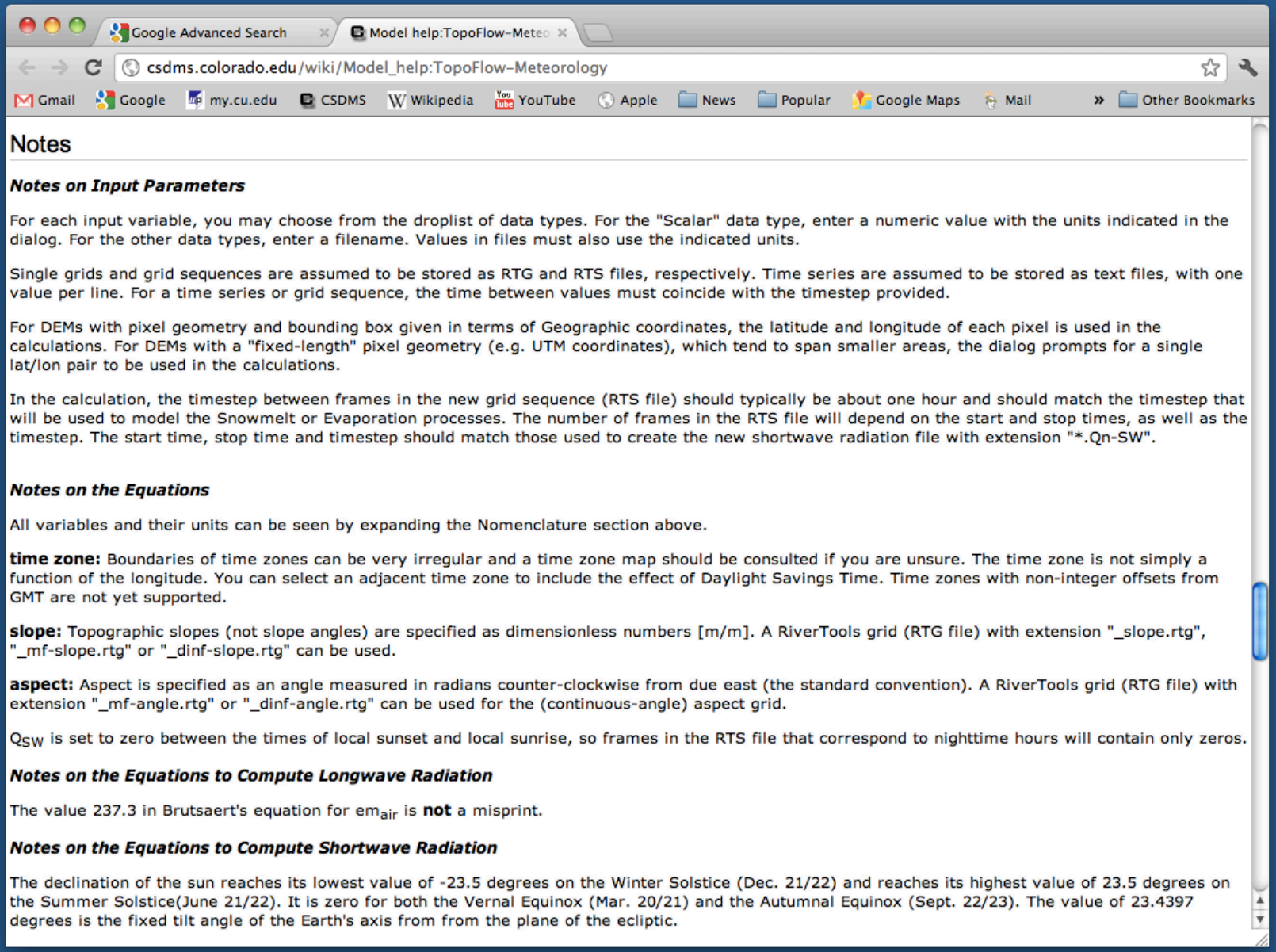
- Saturation vapor pressure, Satterlund (1979) method

$$e_{sat} = \frac{10^{[11.4 - (2353/(T+273.15))]}}{1000} \quad (4)$$

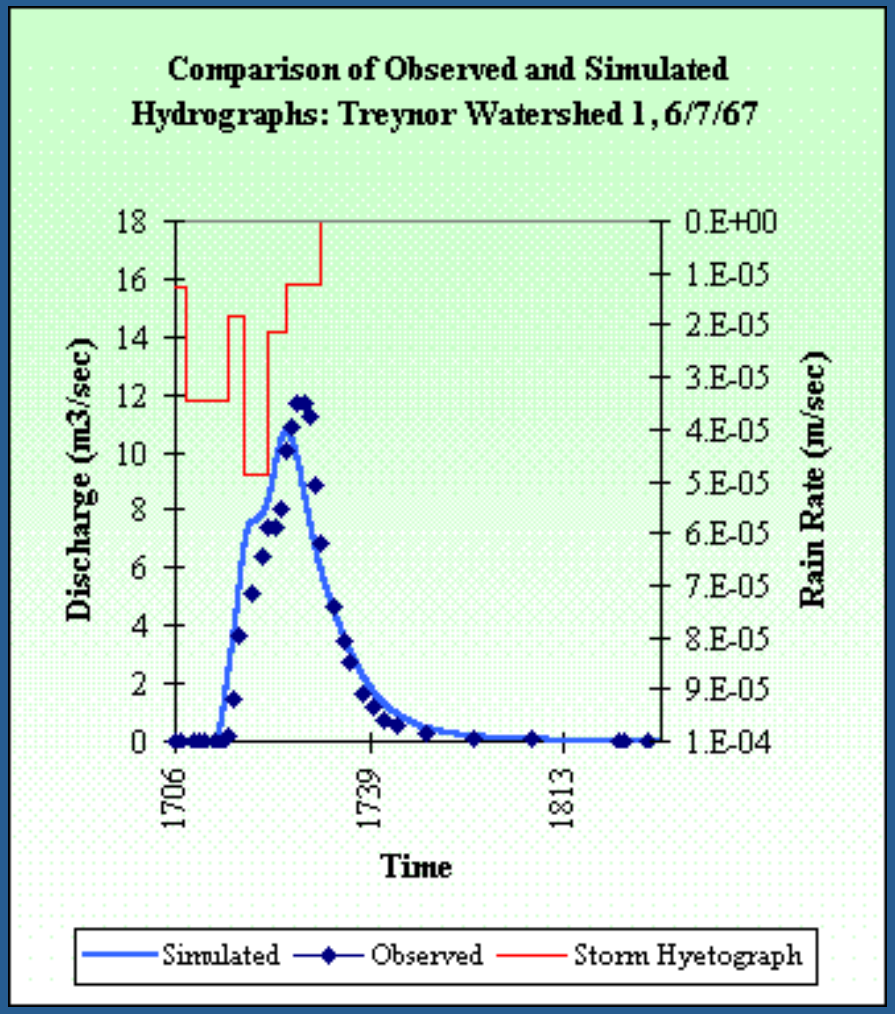
- Vapor pressure of air

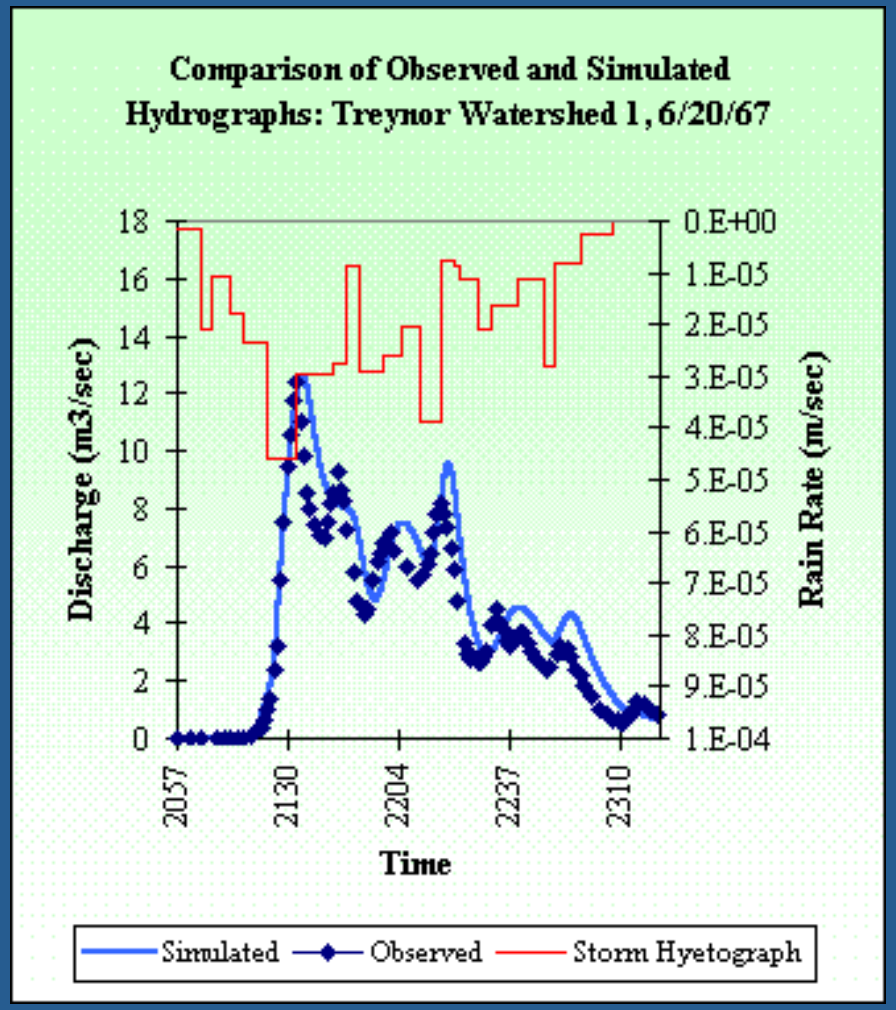
$$e_{air} = e_{sat} \cdot RH \quad (5)$$

HTML Help for Meteorology Component



Modeled vs. Observed Hydrographs for Storms of June 7 and 20, 1967, Treynor Watershed, Iowa





Save Model Results in netCDF Files, Then Use the VisIt Visualization Tool to Make Movies

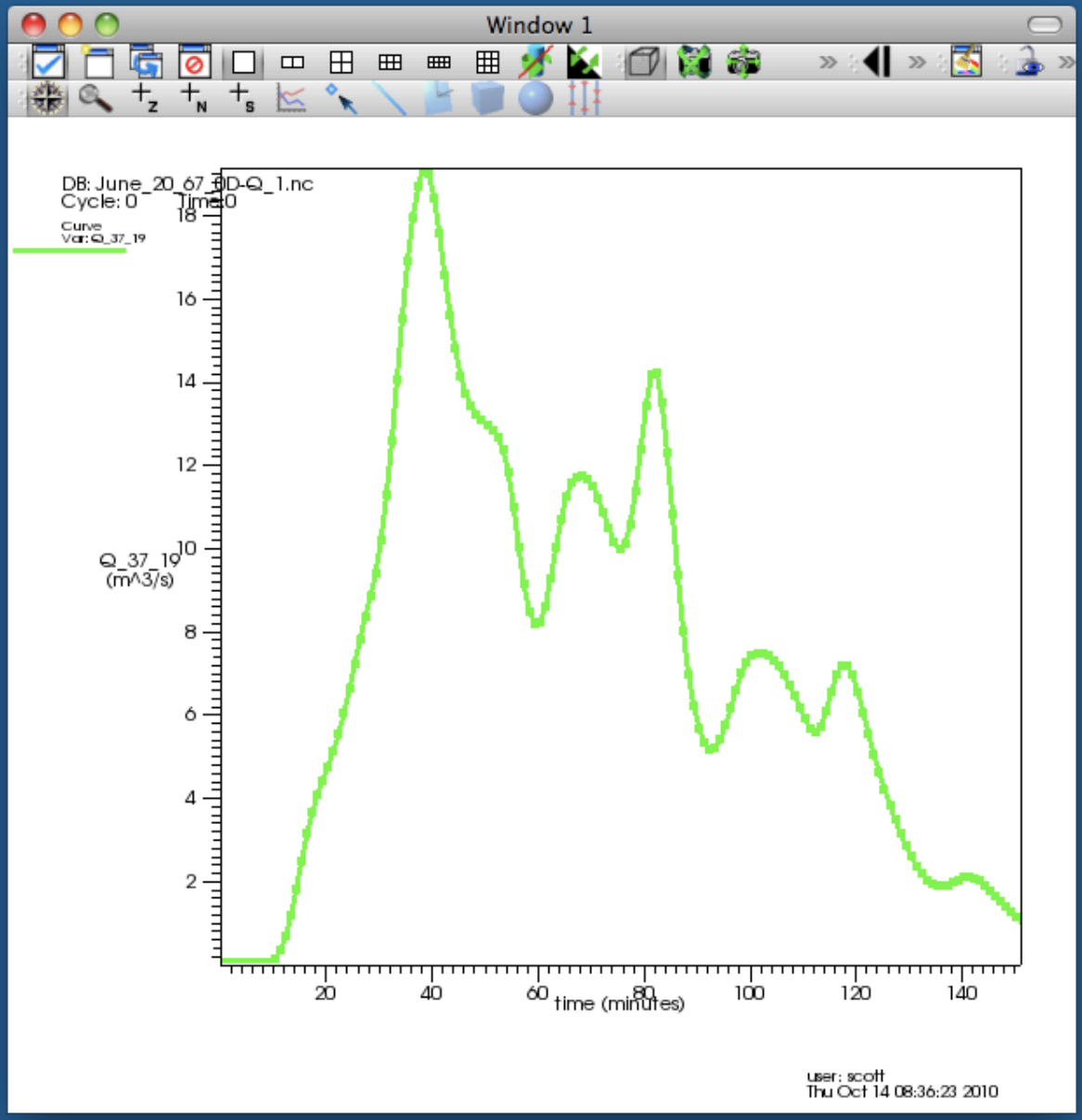
CSDMS Tools for Writing NetCDF

We used the Python package PyNIO to create classes for writing several standard types of output as netCDF files that use the CSDMS Standard Names.

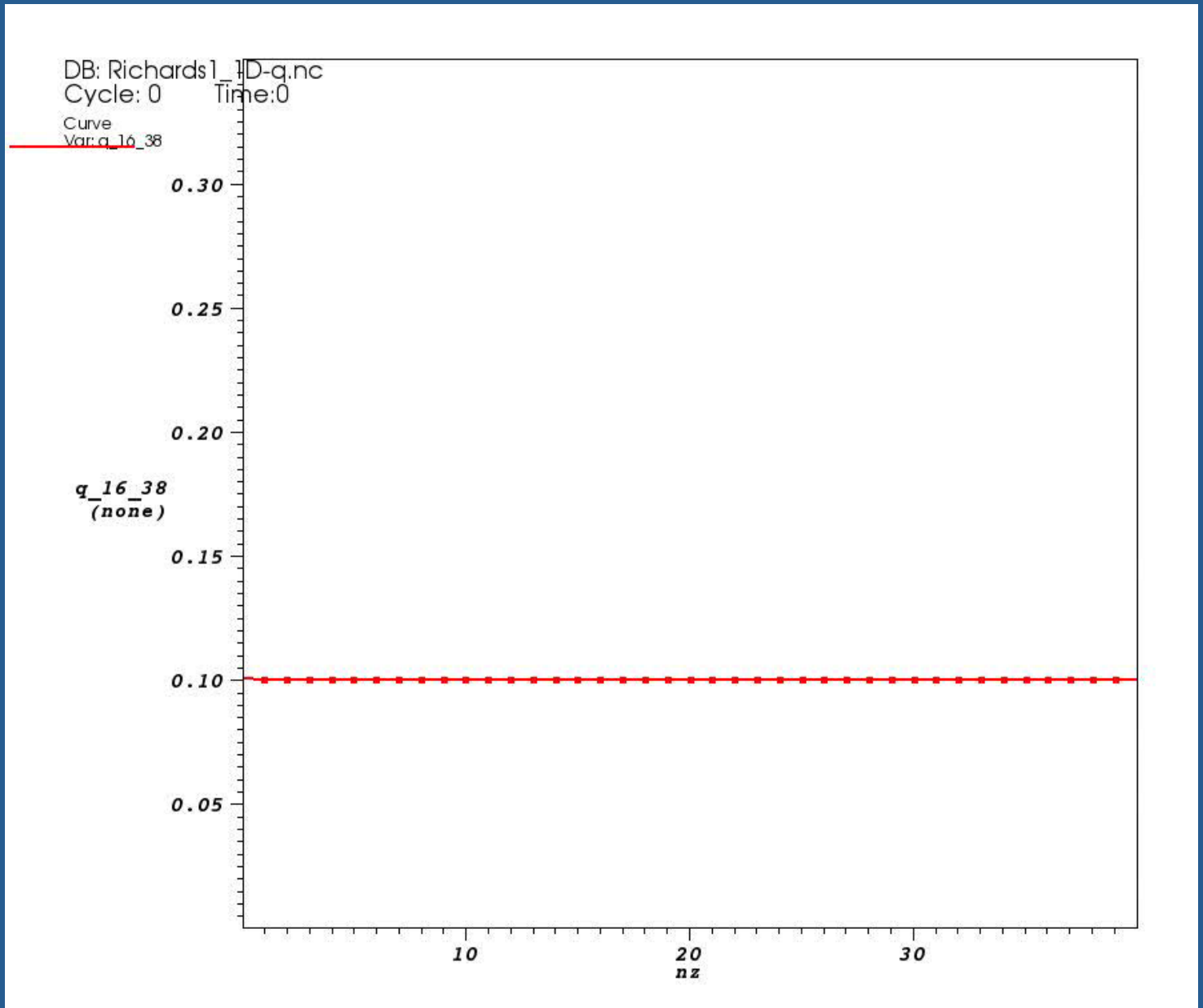
- Time series
- Profile series
- Grid stacks
- Cube stacks
- Scatter plot series (in future)

Regardless of what language your code is written in, CMI can link to this class via Babel to use the tools.

Writing Time Series to NetCDF

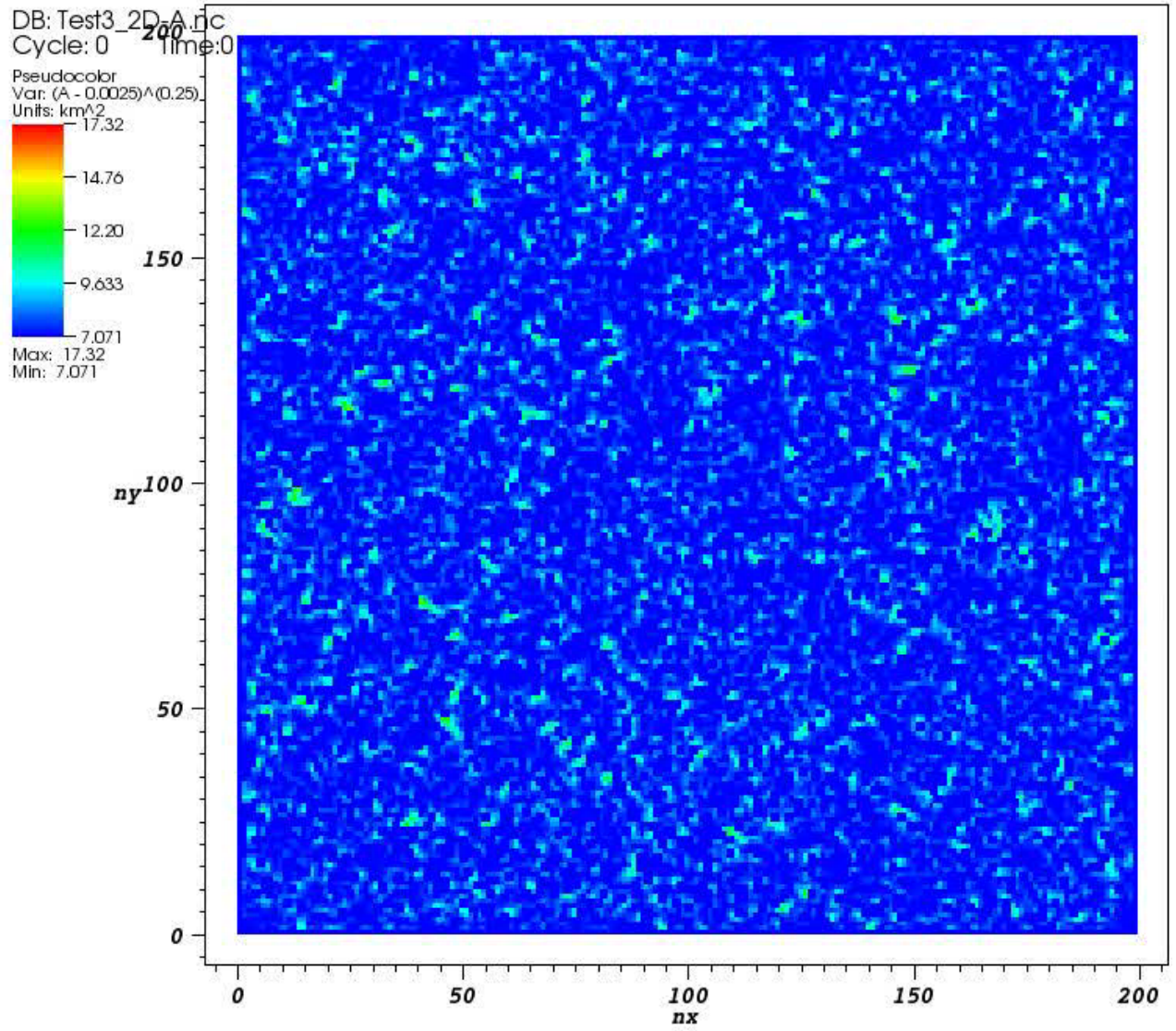


Writing "Profile Stacks" to NetCDF



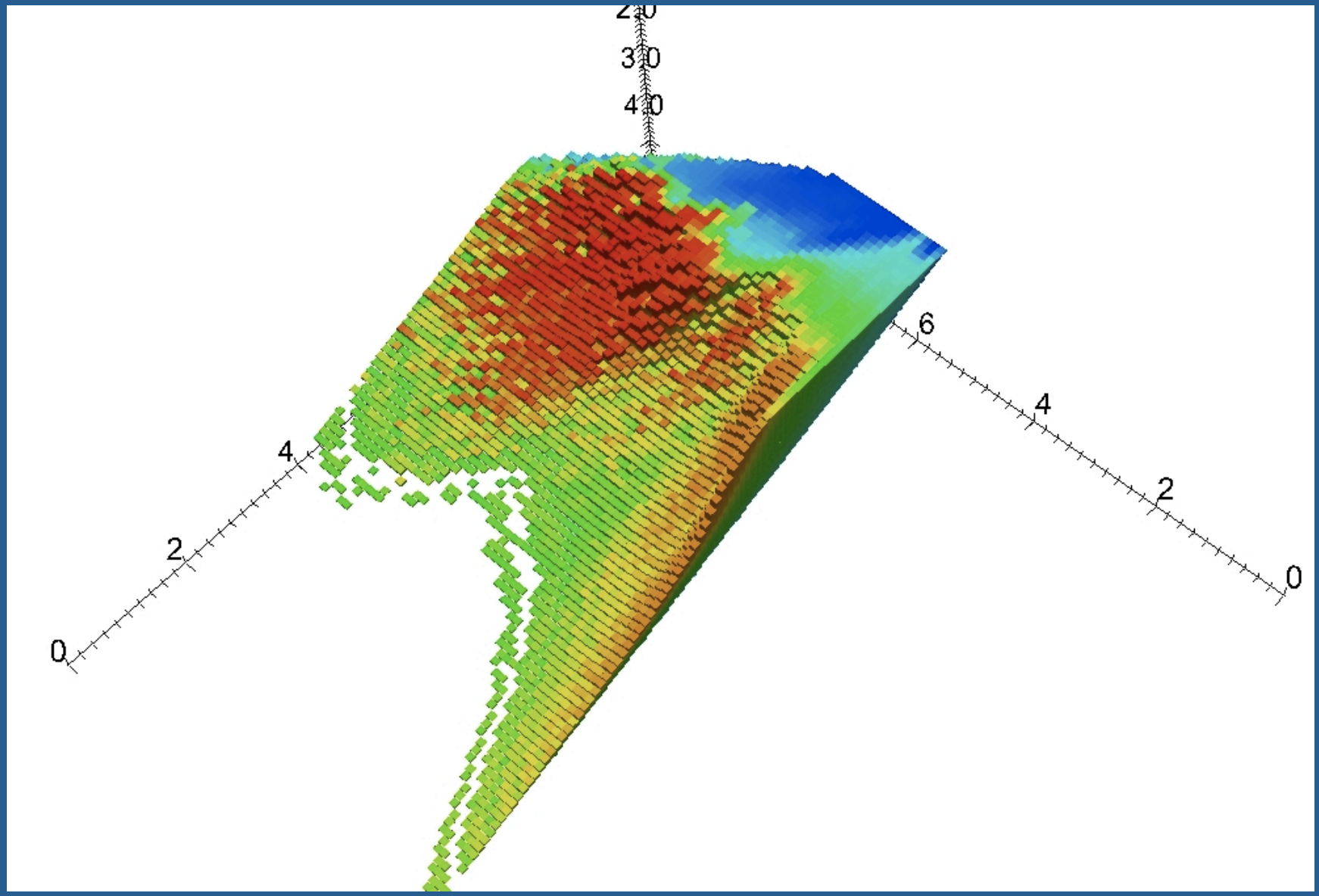
movie

Writing "Grid Stacks" to NetCDF



movie

Writing “Cube Stacks” to NetCDF



Conclusions

- Any CSDMS member can download and install the CSDMS Modeling Tool (CMT) as an easy-to-install Java app.

- Choose model components from available palettes to create customized applications, and run them on our supercomputer. The CMT GUI is linked to VisIt to provide run-time visualization.

- "Process components" are the best level of granularity for model componentization. Processes represent the "scale" at which modelers are most likely to want to replace one approach with another, i.e. to compare different approaches and algorithms with respect to speed, accuracy, scalability or realism.

- Complete models represent the next higher level of granularity but provide limited opportunities for linking and re-use.

More Conclusions

- "Grid stacks" from spatial models can be saved as netCDF files, and then imported into VisIt as a "time-varying" database to allow animation and making movies.

- CSDMS has developed a general set of rules for constructing "CSDMS Standard Names" that address the problem of semantic mediation (model-to-model or model-to-data).

- CSDMS components are designed to be used as either "drivers" (i.e. stand-alone models) or as components.

- Each component has its own GUI for both input and output options as well as its own HTML help page. These pages have many possible uses, such as giving credit to the author of the model component.