

Advanced Use and Concepts

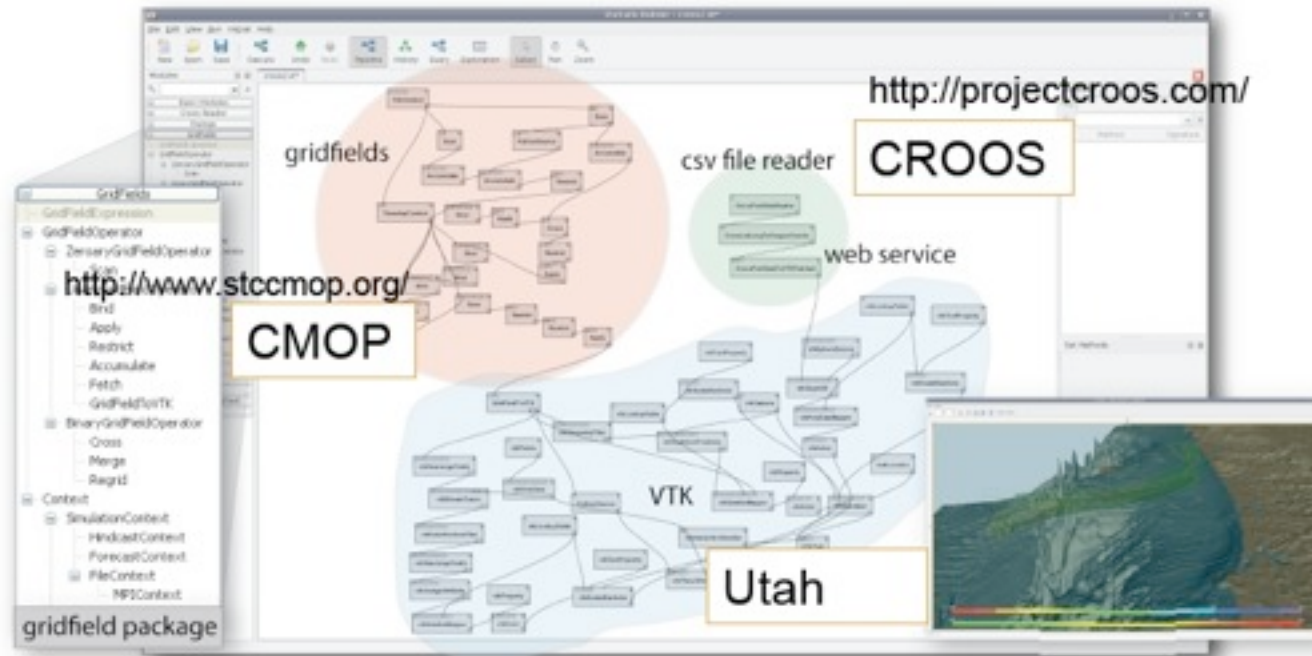
Advanced features of VisTrails

Agenda

- Case-study overview
- Organizational Principles and Practices
- Automatic creation of new workflows
- Advanced Interaction – Part 1
- Advanced Interaction – Part 2
- The workflow debugger

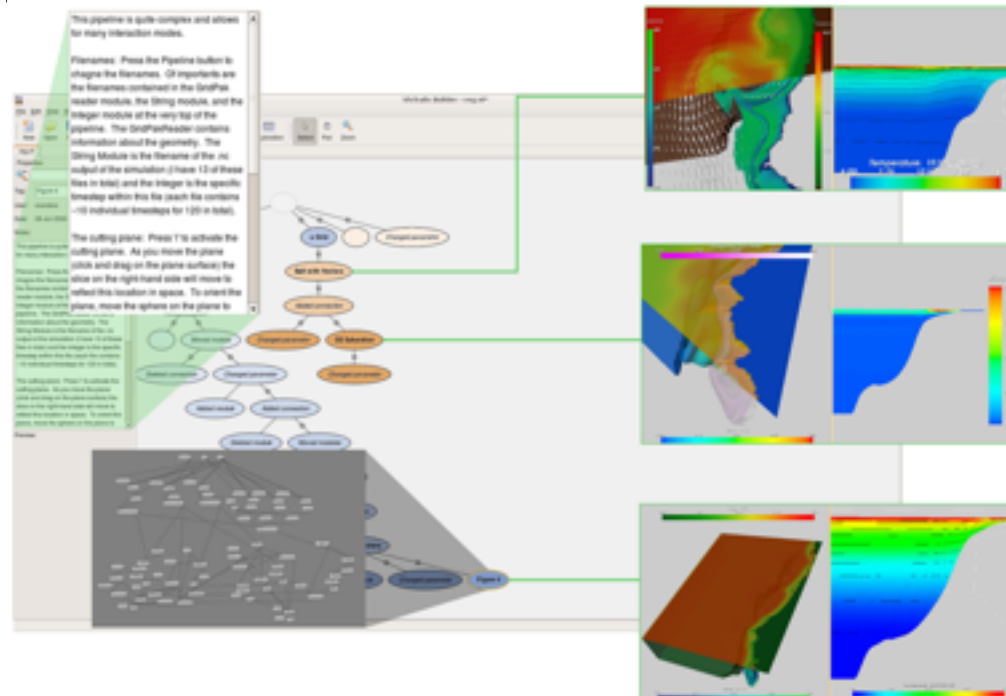
CMOP – A VisTrails example

- Coastal Margin Observation and Prediction
 - Large, multi-modal simulations combined with observational data



CMOP – Provenance enhances collaboration

- Operations and analyses must be recorded and shared
 - Complexity of data and simulation make automatic logging necessary

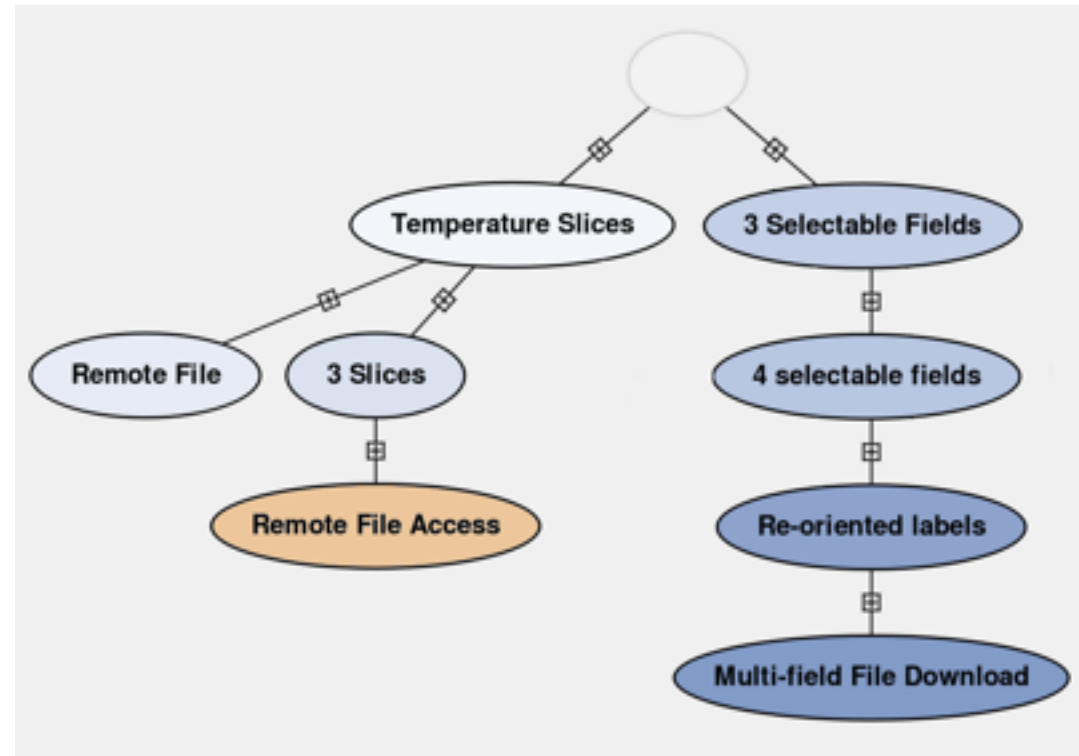


Organizational Principles

Methods of organization for better integration

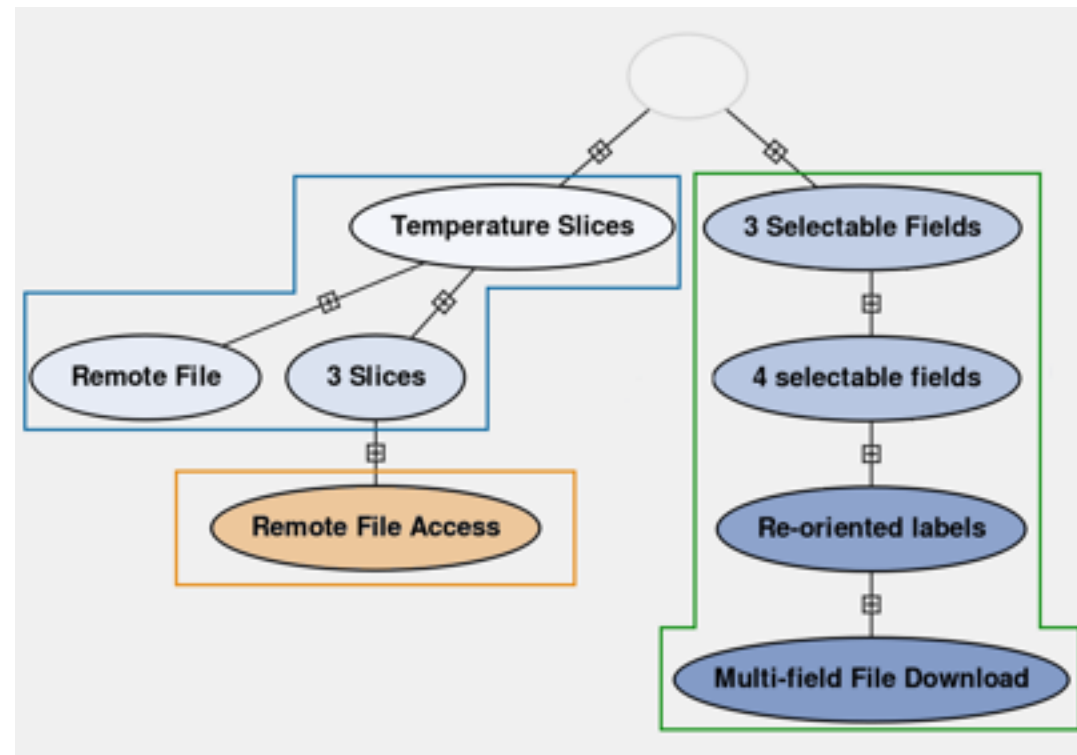
The Version Tree

- Not a “flat” description.
 - Advantageous to use one .vt file for many workflows!



Using the Version Tree

- Proper organization enables
 - Querying
 - Analogies
 - Collaboration



Creation of Workflows

Automatically create complex workflows from simpler building blocks

Demo 1

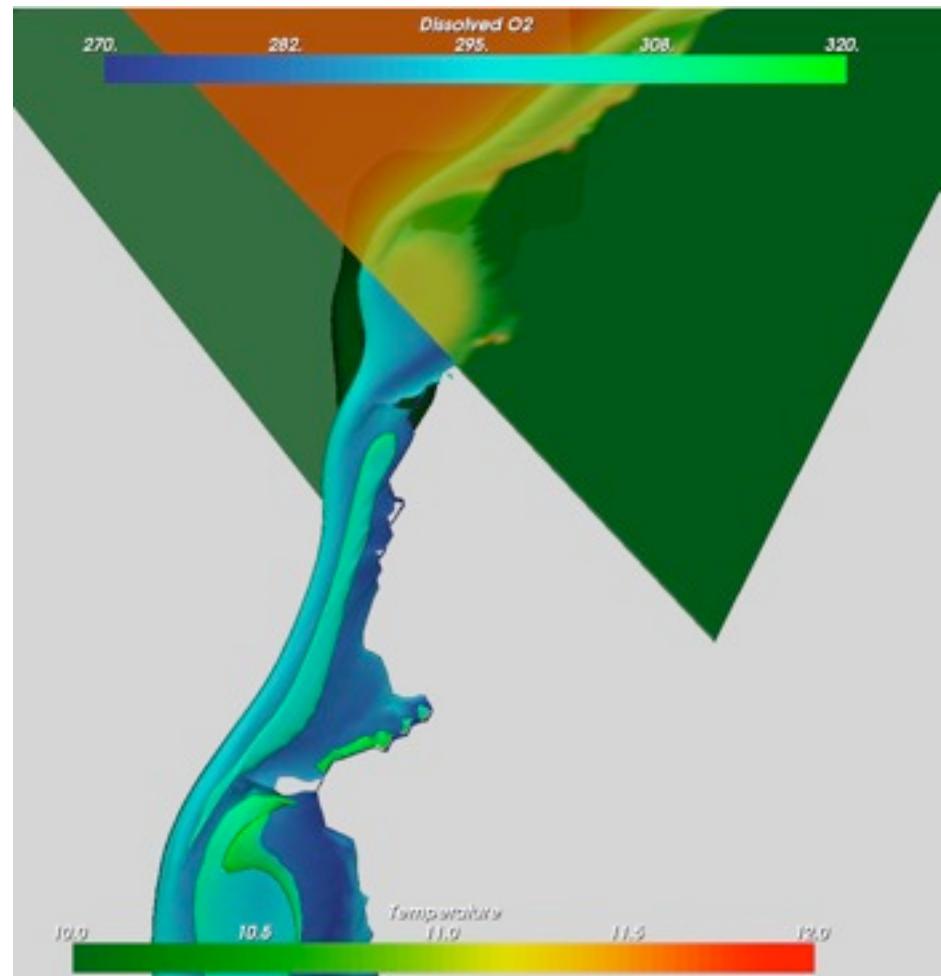
- Using an organized Version Tree to enable searching, querying, and workflow refinement
 - Examples of query-by-example, query-by-user, refinement by parameter exploration, visual diff, refinement by analogy

Advanced Interactions

Enabling VTK Widgets in VisTrails

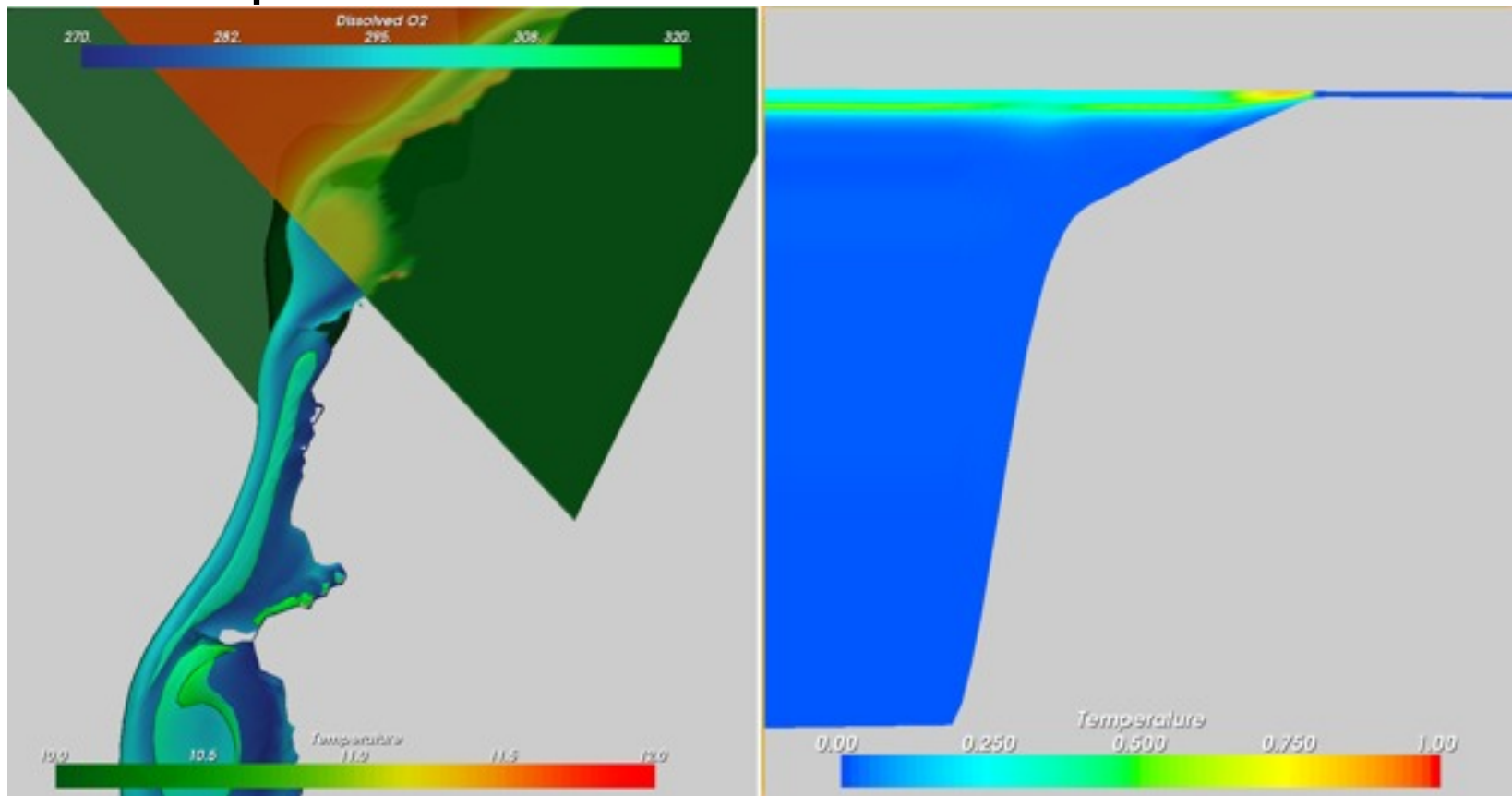
Cutting Planes

- Interaction Handlers
 - Code required to allow specified interactions



Slicing Data

- Multiple Interaction Handlers



Demo 2

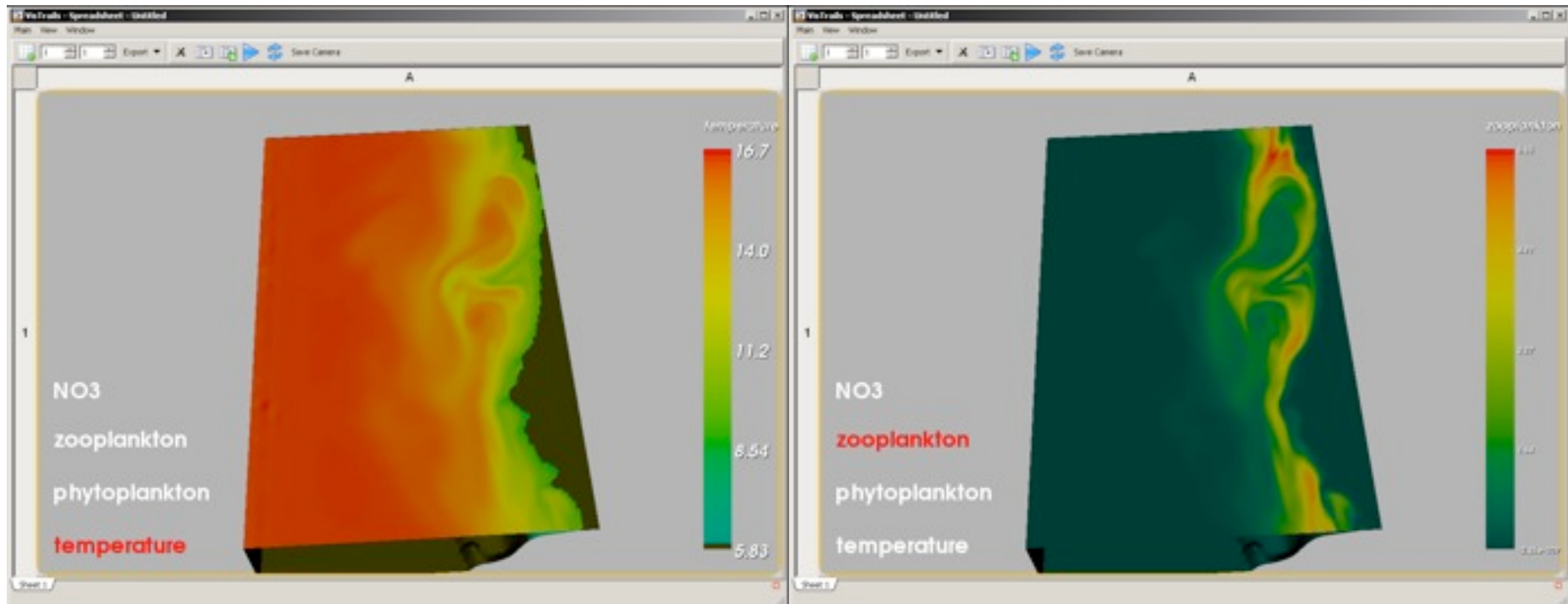
- Supporting basic interaction techniques with `vtkInteractionHandler` Modules
 - Examples of cutting planes, seeding planes, and slicing widgets using the observer pattern in VTK

Advanced Interactions

Enabling interaction through the
callback mechanism

Picking

- Picking is implemented in VTK
 - Any library supporting callbacks uses this mechanism!



Demo 3

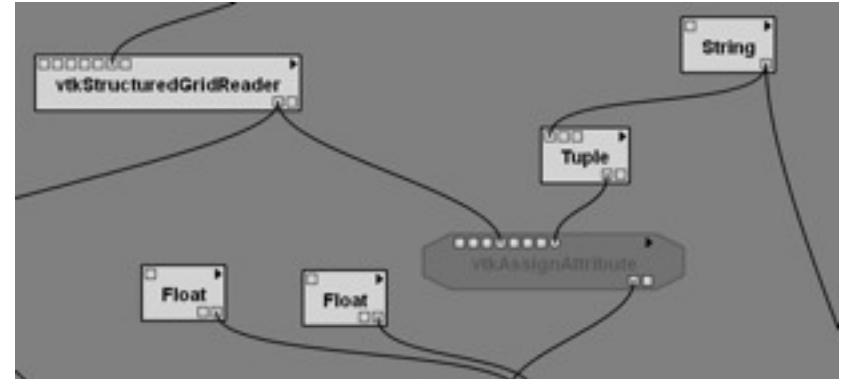
- Supporting advanced interactions with the callback mechanism
 - Example of picking using PythonSource and the VTK callback mechanism

The workflow debugger

Start, pause, and investigate workflows

Debugging

- Breakpoints and watchpoints
 - Workflow analogues to source-code versions



The screenshot shows a software interface with two main panes. The left pane is a console window displaying the following text:

```
Copy Tuple Flags: ( 1 1 1 1 1 0 1 )
DeleteDate Flags: ( 1 1 1 1 1 0 0 )
Pass Through Flags: ( 1 1 1 1 1 1 1 )
Scalars: (none)
Vectors: (none)
Normals: (none)
TCords: (none)
Texors: (none)
GlobalID: (none)
PedigreeID: (none)
Point Data:
Debug: Off
Modified Time: 1257738
Reference Count: 1
Registered Events: (none)
Number Of Arrays: 5
Array 0 name = photoplakton
Array 1 name = velocity
Array 2 name = temperature
Array 3 name = SIO
Array 4 name = copplakton
Number Of Components: 9
Number Of Tuples: 2142688
Copy Tuple Flags: ( 1 1 1 1 1 0 1 )
DeleteDate Flags: ( 1 1 1 1 1 0 0 )
Pass Through Flags: ( 1 1 1 1 1 1 1 )
Scalars:
Debug: Off
```

The right pane is a class browser showing a tree structure of classes. The root is 'vtkScalarVector (3)' with sub-classes 'vtkAssignAttribute (3)' and 'vtkGetOutput'. The 'vtkAssignAttribute (3)' class has a sub-class 'vtkAssignAttribute (3)' with a sub-class 'Assign_2' and a sub-class 'AddInputConnection'. The 'vtkGetOutput' class has a sub-class 'vtkGetOutput'.

Demo 4

- Debugging and inspecting workflows
 - Example of advanced use of the workflow debugger and console. Querying arbitrary module inputs, setting breakpoints and watchpoints.