Environmental Modelling: Crossing Scales and Domains
Bert Jagers

3rd Workshop on Coupling Technologies for Earth System Models
Manchester, April 20-22, 2015
Gold mining has left mercury in the bed of the Sacramento River, so let’s see how these deposits resurface and spread through the delta it shares with the San Joachim River…

… How many model / components would you use?
Domains & Scales
<1m resolution
surface and sewer flows
From local ... Estuary Scale

mixing triangles and quadrangles (as well as pentagons and hexagons were useful)

Delft3D Flexible Mesh
From local ... Coastal Scale
From local ... Sea Scale

Delft3D Flexible Mesh
To global ... South-East Asia
To global ... North America
To global ... Northern Europe

Delft3D Flexible Mesh
To global ... Arctic
Need to dynamically couple components depending on “the research question of the day”

- Geospatial domain different
- Time scales different
- Components different
- Hardware different
... all in a user friendly, GIS based environment
From Engineering (short time scales) ...
... to Geology (long time scales)

Geleynse et al., 2011

HYDRAULIC FORCING TYPE

River
River - Tides
River - Windwaves

PREDOMINANT SUBSURFACE SEDIMENT TYPE

Sand
Sand-Silt
Silt

TU Delft

Deltasres
Scalability

30 km x 1200 m size
2 m x 2 m resolution
= 9 million cells

Delft3D
Storage Intermezzo
Prefer to build on netCDF Climate & Forecasting (CF) conventions

- Rapidly growing adoption

However, it lacks concepts for unstructured meshes

→ New standard: UGRID 0.9


https://github.com/UGRID-conventions
Model Restructuring and Coupling
Starting with FLOW, MOR, and WAVE
- FLOW and MOR run sequentially per time step (parallel model; MPI)
- WAVE may contain multiple nested domains (sequential)
- WAVE runs in parallel (OpenMP or MPI)
- Gridded (2D/3D) quantities exchanged

Extend with CONTROL of hydraulic structures in FLOW
- Scalar quantities exchanged
• We are modifying the Delft3D morphology library to deal with memory management of other host codes.
• MOR contains all Delft3D morphology features such as different sediment transport formulae and bed stratigraphy.
• We are modifying the Delft3D morphology library to deal with memory management of other host codes.
• MOR contains all Delft3D morphology features such as different sediment transport formulae and bed stratigraphy.
Removing MOR from FLOW current solution

Delft3D-FLOW (including MOR)  

Delft3D-FLOW  

Flexible Mesh 1D-2D-3D  

MOSSCO  

Glue layer not drawn to scale

- Glue layer should be standardized as much as possible, like a “coupling” layer
Starting with FLOW (including MOR) and WAVE
- FLOW (MOR) runs in parallel (MPI)
- WAVE may contain multiple nested domains (sequential)
- WAVE runs in parallel (OpenMP or MPI)
- Gridded quantities exchanged

Extend with CONTROL of hydraulic structures in FLOW
- Scalar quantities exchanged
Interaction of FLOW (MOR), WAVE & CONTROL

Vision:

- all memory based data exchange
- a single main exe (coded in C)
- multiple configurable components (coded in Fortran)
Typical exchange fields
• FLOW to WAVE
  • water depth, flow velocities
• WAVE to FLOW
  • wave heights, wave periods, wave forces

Historically implemented as three programs with file based exchange.
Now for Delft3D Flexible Mesh implemented as

- A main program with two libraries
  - FLOW (MOR) component
  - WAVE manager component
- and still a separate WAVE executable.

Data transfer between shared libs initially still via file. Rgrid via ESMF.
Extend with CONTROL as third library component under main.
Data exchange with CONTROL via memory

*MAIN does not know difference between FLOW, WAVE & CONTROL*

*Calling order and data exchange user configured*
Configuration of MAIN

<referenceTime>2015-04-21T00:00:00</referenceTime>
<simulationStartTime>2015-04-21T00:00:00</simulationStartTime>
<simulationEndTime>2016-04-21T00:00:00</simulationEndTime>

<parallel>
  <start name="myFLOWcomponent"/>
  <startGroup>
    <exchangeStartTime>2015-04-21T00:00:00</exchangeStartTime>
    <exchangeTimeStep>PT1H</exchangeTimeStep>
    <exchangeEndTime>2016-04-21T00:00:00</exchangeEndTime>
    <start name="myWAVEcomponent"/>
  </startGroup>
  <startGroup>
    <exchangeStartTime>2015-04-21T00:00:00</exchangeStartTime>
    <exchangeTimeStep>PT1M</exchangeTimeStep>
    <exchangeEndTime>2016-04-21T00:00:00</exchangeEndTime>
    <coupler name="flow2control"/>
    <start name="myCONTROLcomponent"/>
    <coupler name="control2flow"/>
  </startGroup>
</parallel>

<start name="myWAQcomponent"/>

<!-- sequential execution of waq after completion of flow-wave-rtc -->
<start name="myWAQcomponent"/>
<component name="myFLOWcomponent">
  <library>dflowfm</library>
  <workingDir>flowwork</workingDir>
  <process>0 1</process>
  <!-- Additional(component specific) keywords. -->
  <inputFile>input/flow/r17.mdu</inputFile>
  <partitionFile>partitions.nc</partitionFile>
</component>

<component name="myWAVEcomponent">
  <library>wave</library>
  <workingDir>wavework</workingDir>
  <process>0 1 2 3</process>
  <!-- Additional(component specific) keywords. -->
  <inputFile>input/wave/r17.mdw</inputFile>
</component>
<coupler name="flow2control">
  <sourceComponent>myFLOWcomponent</sourceComponent>
  <targetComponent>myCONTROLcomponent</targetComponent>
  <item>
    <sourceName>observations/Obs_A/water_level</sourceName>
    <targetName>Pid_Controller_A/h</targetName>
    <!-- Possibility for adapter arguments here in the future -->
  </item>
  <!-- more exchange items -->
</coupler>
IRF + get/set + self-describing (introspection) … see Scott’s talk

```plaintext
integer(c_int) function initialize(c_config_file) result(c_iresult) bind(C, name="initialize")
integer function update(dt) bind(C, name="update")
integer function finalize() bind(C, name="finalize")

subroutine get_start_time(t) bind(C, name="get_start_time")
subroutine get_end_time(t) bind(C, name="get_end_time")
subroutine get_time_step(dt) bind(C, name="get_time_step")
subroutine get_current_time(t) bind(C, name="get_current_time")

subroutine get_var_type(c_var_name, c_type) bind(C, name="get_var_type")
subroutine get_var_rank(c_var_name, rank) bind(C, name="get_var_rank")
subroutine get_var_shape(c_var_name, shape) bind(C, name="get_var_shape")
subroutine get_var_count(c_var_count) bind(C, name="get_var_count")
subroutine get_var_name(var_index, c_var_name) bind(C, name="get_var_name")
subroutine get_var(c_var_name, x) bind(C, name="get_var")

subroutine set_var(c_var_name, xptr) bind(C, name="set_var")
```

grid information for spatial interpolation
API requirements (aiming for BMI update)

IRF + get/set + self-describing (introspection) … see Scott’s talk

integer(c_int) function initialize(c_config_file) result(c_iresult) bind(C, name="initialize")
integer function update(dt) bind(C, name="update")
integer function finalize() bind(C, name="finalize")

subroutine get_start_time(t) bind(C, name="get_start_time")
subroutine get_end_time(t) bind(C, name="get_end_time")
subroutine get_time_step(dt) bind(C, name="get_time_step")
subroutine get_current_time(t) bind(C, name="get_current_time")
subroutine get_var_type(c_var_name, c_type) bind(C, name="get_var_type")
subroutine get_var_rank(c_var_name, rank) bind(C, name="get_var_rank")
subroutine get_var_shape(c_var_name, shape) bind(C, name="get_var_shape")
subroutine get_var_count(c_var_count) bind(C, name="get_var_count")
subroutine get_var_name(var_index, c_var_name) bind(C, name="get_var_name")
subroutine get_var(c_var_name, x) bind(C, name="get_var")
subroutine set_var(c_var_name, xptr) bind(C, name="set_var")

Multi level initialize and finalize needed

simple: water_level
multi level: observations -> Obs_A -> water_level

Risk time consuming parsing of strings to get a single value. Use something like an “id” as alternative? Unfortunately this would imply buffering of references at the component side, so maybe this should be optional.

Parallel version (no gather)

grid information for spatial interpolation

Need a bit more flexibility than BMI currently offers
Coupling
Variations on a Theme
Multiple instances of the same model

Different groups responsible for different parts of the national model.

Rather than one model, maintain separate models that can be coupled.

… all together coupled to a national ground water model (almost via OpenMI)
Interactive modelling

Interact with the model while it’s running.

“You are one of the coupled components.”

https://www.youtube.com/watch?v=kzFBZ8KaWF4
Web based modelling

- Stateless:
  - Pull driven
  - query \rightarrow answer

- Stateful:
  - Publish & subscribe pattern
  - query a running process

- Increasing interactiveness

- event driven

Deltabes
Model Message Interface (MMI)
https://www.nuget.org/packages/ModelMessageInterface/

- Built on top of Basic Model Interface
  http://csdms.colorado.edu/wiki/BMI_Description

- JSON message ({"name": "waterlevel", "shape": [300, 200]}) + Binary Message (Array)

- Built on top of message passing code/protocol ØMQ:
  http://zeromq.org/
Model implements
• Initialize, Timestep ("Run"), Finalize
• Get / Set functions – Introspection

The model does not have to call any other component: the wrapper will call you (Hollywood principle).

Similar wrappers can be implemented for ESMF, OpenMI, CCA, and other frameworks.
Asynchronous remote control

- Visualizer
- Controller

→ query
→→ event trigger

MMI wrapper

Model
BMI interface