Linking Arbitrary Hydrologic and Hydraulic Models and Process Modules into a Single Prediction Runtime Using the Basic Model Interface: A Domain Science Perspective

Presenter: Jessica L. Garrett

Lynker

Jessica L. Garrett[1,3], Robert J. Bartel[1], Shengting T. Cui[1,2], Luciana L. Kindl da Cunha[1,6], Trey C. Flowers[1], Jonathan M. Frame[1,5], Nels J. Frazier[1,2], Keith S. Jennings[1,3], Fred L. Ogden[1], Scott D. Peckham[1,4]

1. NOAA, National Water Center 2. ERT, Inc. 3. Lynker Technologies 4. University of Colorado 5. University of Alabama 6. WEST Consultant, Inc.

What is a BMI

Basic Model Interface:

*A set of essential, "self-describing", functions used to **control** (e.g. advance-of-time) & **access** (i.e. query) the state of a model*

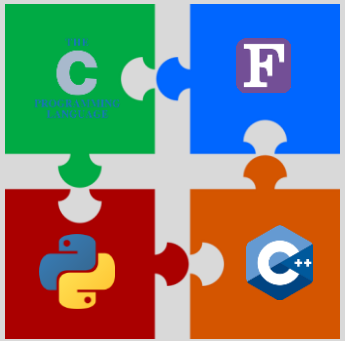
- Common Interface: Information exchange
- Flexibility:
  - Versioning
  - Can be wrapped to other externals APIs
- Interoperability
  - Programming languages: C, C++, Python and Fortran
  - Extendable as a *data* component also

    see: https://csdms.colorado.edu/wiki/DataComponents
- Non-invasive
  - Model predictions remain untouched
  - Runtime performance is minimally affected
- Community
  - Names and units are standardized
  - A simplified implementation; compared to other similar APIs

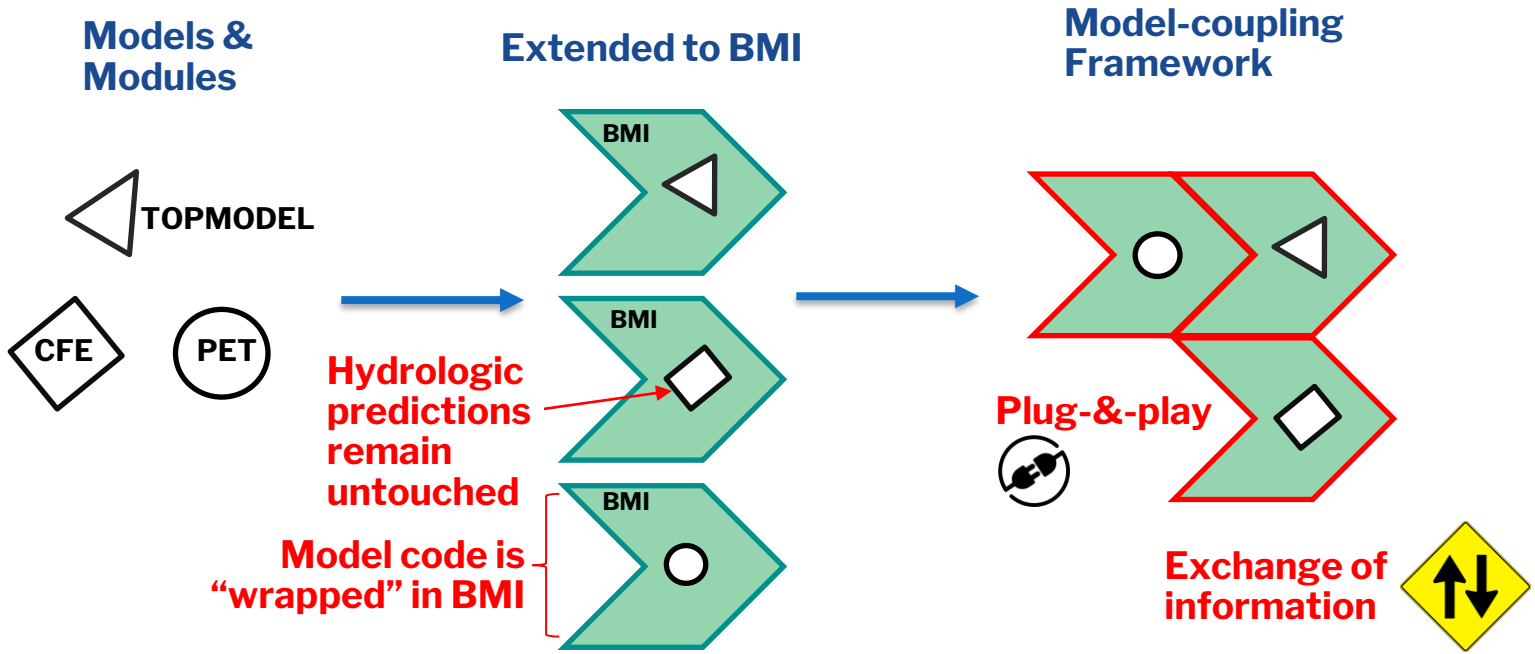
bmi

Developed by: **Community Surface Dynamics Modeling System (CSDMS)**

The Perks



How a BMI Works



Adaption: Getting Started

Q: How to wrap an existing model with BMI?
A: Start with function definitions

Model Control (4)
Model Information (5)
Model Time (5)
Model Grid (16)
Variable Information (6)
Variable Getters & Setters (5)

More how-tos and documentation provided via CSDMS:
https://csdms.colorado.edu/wiki/BMI

```c
typedef struct Bmi {
    void *data;

    /* Initialize, run, finalize (IRF) */
    int (*initialize)(struct Bmi *self, const char *config_file);
    int (*update)(struct Bmi *self);
    int (*update_until)(struct Bmi *self, double then);
    int (*finalize)(struct Bmi *self);

    /* Exchange items */
    int (*get_component_name)(struct Bmi *self, char *name);
    int (*get_input_item_count)(struct Bmi *self, int *count);
    int (*get_output_item_count)(struct Bmi *self, int *count);
    int (*get_input_var_names)(struct Bmi *self, char **names);
    int (*get_output_var_names)(struct Bmi *self, char **names);

    /* Variable information */
    int (*get_var_grid)(struct Bmi *self, const char *name, int *grid);
    int (*get_var_type)(struct Bmi *self, const char *name, char *type);
    int (*get_var_units)(struct Bmi *self, const char *name, char *units);
    int (*get_var_itemsize)(struct Bmi *self, const char *name, int *size);
    int (*get_var_nbytes)(struct Bmi *self, const char *name, int *nbytes);
    int (*get_var_location)(struct Bmi *self, const char *name, char *location);

    /* Time information */
    int (*get_current_time)(struct Bmi *self, double *time);
    int (*get_start_time)(struct Bmi *self, double *time);
    int (*get_end_time)(struct Bmi *self, double *time);
    int (*get_time_units)(struct Bmi *self, char *units);
    int (*get_time_step)(struct Bmi *self, double *time_step);

    /* Getters and setters */
    int (*get_value)(struct Bmi *self, const char *name, void *dest);
    int (*get_value_ptr)(struct Bmi *self, const char *name, void **dest_ptr);
    int (*get_value_at_indices)(struct Bmi *self, const char *name, void *dest, int *inds, int count);
    int (*set_value)(struct Bmi *self, const char *name, void *src);
    int (*set_value_at_indices)(struct Bmi *self, const char *name, int *inds, int count, void *src);

    /* Grid information */
    int (*get_grid_rank)(struct Bmi *self, int grid, int *rank);
    int (*get_grid_size)(struct Bmi *self, int grid, int *size);
    int (*get_grid_type)(struct Bmi *self, int grid, char *type);

    /* Uniform rectilinear */
    int (*get_grid_shape)(struct Bmi *self, int grid, int *shape);
    int (*get_grid_spacing)(struct Bmi *self, int grid, double *spacing);
    int (*get_grid_origin)(struct Bmi *self, int grid, double *origin);

    /* Non-uniform rectilinear, curvilinear */
    int (*get_grid_x)(struct Bmi *self, int grid, double *x);
    int (*get_grid_y)(struct Bmi *self, int grid, double *y);
    int (*get_grid_z)(struct Bmi *self, int grid, double *z);

    /* Unstructured */
    int (*get_grid_node_count)(struct Bmi *self, int grid, int *count);
    int (*get_grid_edge_count)(struct Bmi *self, int grid, int *count);
    int (*get_grid_face_count)(struct Bmi *self, int grid, int *count);
    int (*get_grid_edge_nodes)(struct Bmi *self, int grid, int *edge_nodes);
    int (*get_grid_face_edges)(struct Bmi *self, int grid, int *face_edges);
    int (*get_grid_face_nodes)(struct Bmi *self, int grid, int *face_nodes);
    int (*get_grid_nodes_per_face)(struct Bmi *self, int grid, int *nodes_per_face);
} Bmi;
```

```
// ************************************************************
// ************** BMI: MODEL CONTROL FUNCTIONS ****************
// ************************************************************

static int Initialize (Bmi *self, const char *cfg_file)
{
    topmodel_model *topmodel;
    topmodel = (topmodel_model *) self->data;

    // Read and setup data from file
    init_config(cfg_file, topmodel);

    // Initialize model varables
    topmodel->current_time_step=0;
    topmodel->sump = 0.0;
    topmodel->sumae = 0.0;
    topmodel->sumq = 0.0;

    return BMI_SUCCESS;
}

static int Update (Bmi *self)
{
    topmodel_model *topmodel;
    topmodel = (topmodel_model *) self->data;

    double current_time, end_time;
    self->get_current_time(self, &current_time);
    self->get_end_time(self, &end_time);
    if (current_time >= end_time) {
        return BMI_FAILURE;
    };

    topmodel->current_time_step += topmodel->dt;

    // call model run (1 timestep)
    topmod(topmodel->output_fptr,topmodel->nstep, topmodel->num_topodex_values, ...

    return BMI_SUCCESS;
}

static int Finalize (Bmi *self)
{
  if (self){
    topmodel_model* model = (topmodel_model *)(self->data);

    water_balance(model->output_fptr, model->yes_print_output, ...

    results(topmodel->output_fptr, topmodel->out_hyd_fptr, topmodel->nstep, ...

    free(self->data);
  }
    return BMI_SUCCESS;
}
```

Adaption: Control Functions

bmi.initialize():

- **Reads a configuration file**
- Open/close files
- Allocates memory
- Sets initial data values

bmi.update():

- Advances model state
- Iterates clocktime

bmi.finalize():

- Frees memory
- Computes end-of-run summaries

```c
// ************************************************************
// ****************** BMI: TIME FUNCTIONS *********************
// ************************************************************

static int Get_start_time (Bmi *self, double * time)
{
    *time = 0.0;
    return BMI_SUCCESS;
}

static int Get_end_time (Bmi *self, double * time)
{

    topmodel_model *topmodel;
    topmodel = (topmodel_model *) self->data;
    Get_start_time(self, time);

    // Standalone case gathers end_time via nstep dt (from file)
    if (topmodel->stand_alone == TRUE){
        *time += topmodel->nstep * topmodel->dt;
        return BMI_SUCCESS;

    // Otherwise, set to FLT_MAX macro via float.h
    // See https://bmi.readthedocs.io/en/latest/#get-end-time
    }
    else {
        *time += FLT_MAX;
        return BMI_SUCCESS;
    }
}

static int Get_time_step (Bmi *self, double * dt)
{
    *dt = ((topmodel_model *) self->data)->dt;
    return BMI_SUCCESS;
}

static int Get_time_units (Bmi *self, char * units)
{
    strncpy (units, "h", BMI_MAX_UNITS_NAME);
    return BMI_SUCCESS;
}

static int Get_current_time (Bmi *self, double * time)
{
    Get_start_time(self, time);
    *time += (((topmodel_model *) self->data)->current_time_step *
              ((topmodel_model *) self->data)->dt);
    return BMI_SUCCESS;
}
```

Adaption: Time Functions

bmi.get_start_time(): typically returns `0`

bmi.get_end_time():

- Likely driven via model-coupling framework
- ∴ set some MAX (default safety-catch)
- But consider a file-read-in value too...

bmi.get_time_step(): *size* or *delta* - likely set to `1`

bmi.get_time_units():

- Units: **exchange item**
- Must be standardized

UDUNITS: C Package for naming & numerical conversions for physical units
https://www.unidata.ucar.edu/software/udunits/

bmi.get_current_time (): counter++

```c
#define INPUT_VAR_NAME_COUNT 2

static const char *input_var_names[INPUT_VAR_NAME_COUNT] = {
        "atmosphere_water__liquid_equivalent_precipitation_rate",
        "water_potential_evaporation_flux"
};
static const char *input_var_units[INPUT_VAR_NAME_COUNT] = {
        "m h-1",
        "m h-1"
};
static const char input_var_grids[INPUT_VAR_NAME_COUNT] = {
        0,
        0
};

static int Get_input_var_names (Bmi *self, char ** names)
{
    for (int i = 0; i < INPUT_VAR_NAME_COUNT; i++) {
        strncpy (names[i], input_var_names[i], BMI_MAX_VAR_NAME);
    }
    return BMI_SUCCESS;
}

static int Get_var_units (Bmi *self, const char *name, char * units)
{
    // Check to see if in output array first
    for (int i = 0; i < OUTPUT_VAR_NAME_COUNT; i++) { ...
    }
    // Then check to see if in input array
    for (int i = 0; i < INPUT_VAR_NAME_COUNT; i++) {
        if (strcmp(name, input_var_names[i]) == 0) {
            strncpy(units, input_var_units[i], BMI_MAX_UNITS_NAME);
            return BMI_SUCCESS;
        }
    }
    // If we get here, it means the variable name wasn't recognized
    return BMI_FAILURE;
}

static int Get_grid_type (Bmi *self, int grid, char * type)
{
    int status = BMI_FAILURE;

    // grid id = 0 ... type scalar NOT A TRUE GRIDED MODEL
    if (grid == 0) {
        strncpy(type, "scalar", BMI_MAX_TYPE_NAME);
        status = BMI_SUCCESS;
    }
    else {
        type[0] = '\0';
        status = BMI_FAILURE;
    }
    return status;
}

/* Uniform rectilinear (grid type) */
static int Get_grid_shape(Bmi *self, int grid, int *shape)
{
    return BMI_FAILURE;
}
```

Adaption: Information Functions

bmi.get_input_var_names() & bmi.get_output_var_names:

- Names: **exchange item**
- Must be standardized

https://csdms.colorado.edu/wiki/CSDMS_Standard_Names

bmi.var_units():

- Units: **exchange item**
- Must be standardized

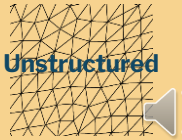
UDUNITS: C Package for naming & numerical conversions for physical units
https://www.unidata.ucar.edu/software/udunits/

BMI Grid Functions (16) :

- Implementation & definition depends on model grid
- If not a true gridded model, then,
  - Type: `scalar`
  - Rank: `1`
  - Size: `1`
- Support for gridded models:
  - Uniform
  - Rectangular
  - Structured
  - Unstructured

3 functions
Scalar

6 functions


16 functions


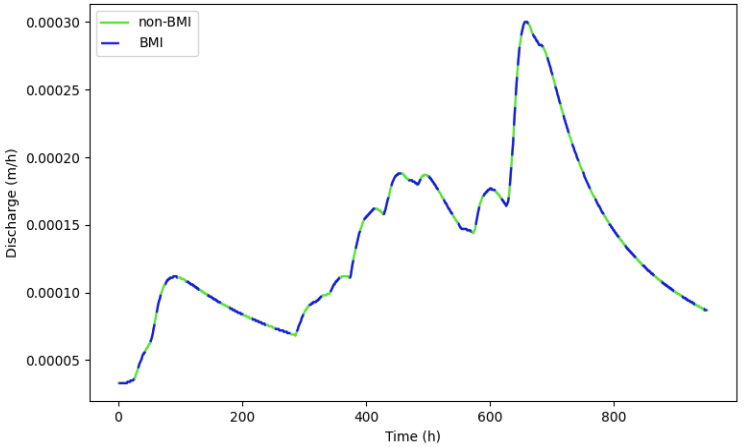
```
for (i=0; i<count_in; i++){
    const char *var_name = names_in[i];
    printf( " %s\n", var_name);
    // Test get_var_grid()
    {
        status = model->get_var_grid(model, var_name, &grid);
        if (status == BMI_FAILURE) return BMI_FAILURE;
        printf( "  grid: %i\n", grid);
    }
    // Test get_var_itemsize()
    {
        status = model->get_var_itemsize(model, var_name, &itemsize);
        if (status == BMI_FAILURE) return BMI_FAILURE;
        printf( "  itemsize: %i\n", itemsize);
    }
    { // Test get_var_location()
        status = model->get_var_location(model, var_name, location);
        if (status == BMI_FAILURE) return BMI_FAILURE;
        printf( "  location: %s\n", location);
    }
    // Test get_var_units()
    {
        status = model->get_var_units(model, var_name, units);
        if (status == BMI_FAILURE) return BMI_FAILURE;
        printf( "  units: %s\n", units);
    }
    // Test get_var_type()
    {
        status = model->get_var_type(model, var_name, type);
        if (status == BMI_FAILURE) return BMI_FAILURE;
        printf( "  type: %s\n", type);
    }
    { // get_var_nbytes()
        status = model->get_var_nbytes(model, var_name, &nbytes);
        if (status == BMI_FAILURE) return BMI_FAILURE;
        printf( "  nbytes: %i\n", nbytes);
    }
}
```

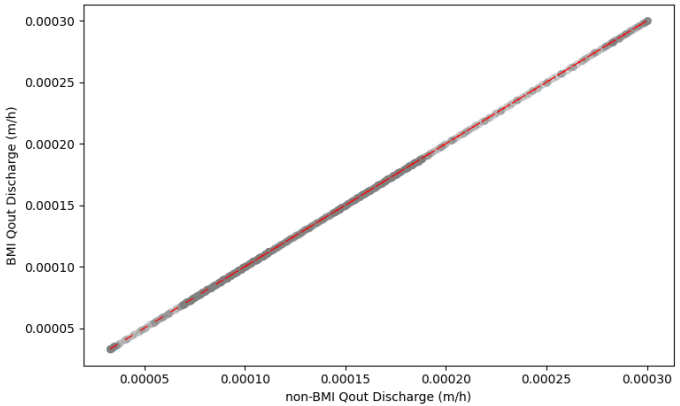
Adaption: Unit Tests

```
BEGIN BMI UNIT TEST
*******************
 allocating memory for model structure...
 registering BMI model...
 initializing... configuration found: ./data/topmod_100.run

MODEL INFORMATION FUNCTIONS
***********************************
 component name: TOPMODEL
 input item count: 2
 output item count: 14

VARIABLE INFORMATION FUNCTIONS
****************************************
 atmosphere_water__liquid_equivalent_precipitation_rate
  grid: 0
  itemsize: 8
  location: node
  units: m h-1
  type: double
  nbytes: 8

TIME FUNCTIONS
***************
 start time: 0.0
 current time: 0
 time step size: 1
 time step units: 1 hour

GRID INFORMATION
****************
 grid id: 0
  rank: 1
  size: 1
  type: scalar

CONTROL FUNCTIONS
*****************
 updating...        time 1
 updating until...  time 100
 finalizing...

 Total BMI function PASS: 28
 Total BMI function FAIL: 0
```

Adaption: Test for Side Effects

- Runtime Performance
  - Minimal computation cost
  - Expected values – results unchanged
- Model Code:
  - Minimal adjustments to primary function definitions
  - Maintains original internal names and units
  - Still recognizable by original author/developer?





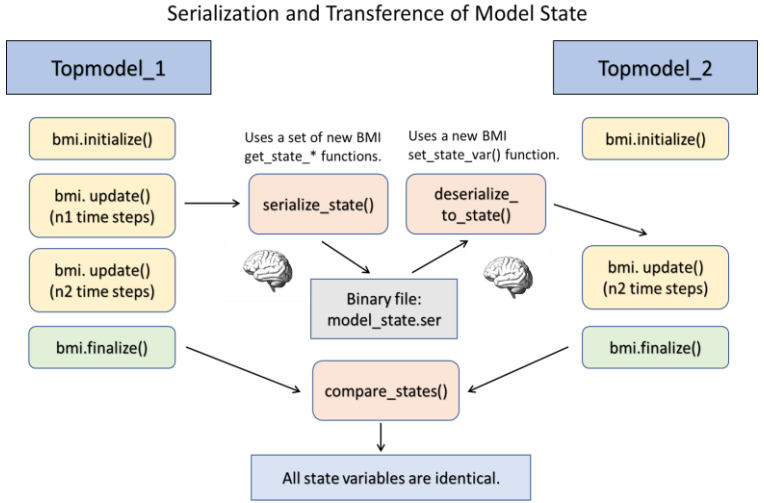
TOPMODEL: Comparing BMI Outputs

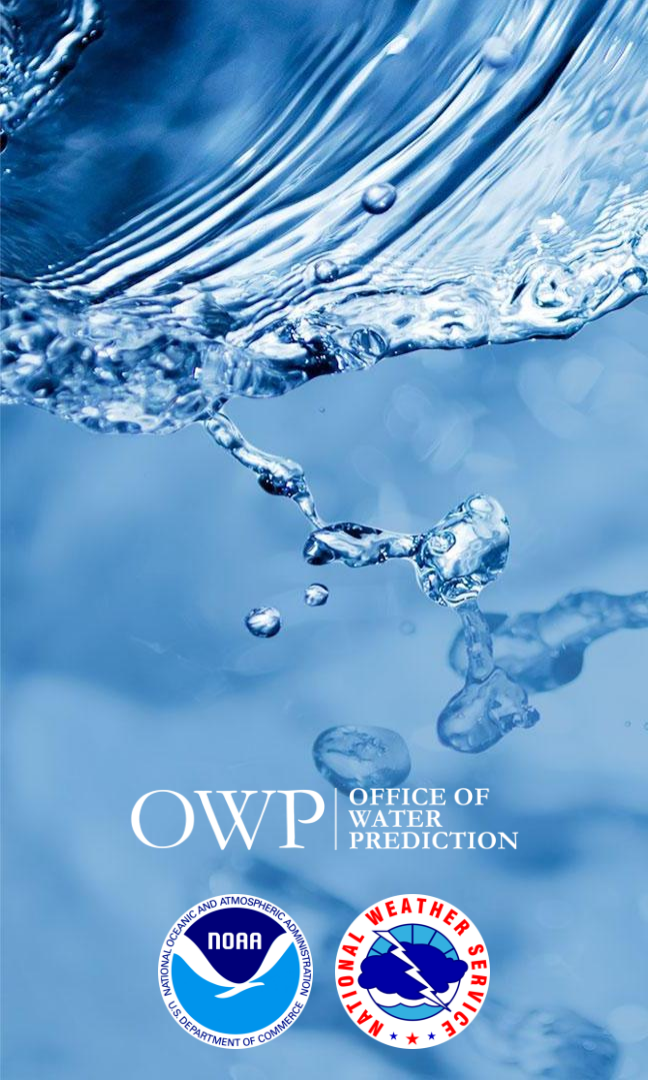
Adaption: Development Notes

- Domain Scientist: BMI control functions
- All functions return either
  - BMI_SUCCESS
  - BMI_FAILURE (if not implemented, e.g. model grid)
- Remove “hard-coded” time loop
- Stdout = Low verbosity
- Mindful of when files are opened/closed
- Use standard variable names and units
- Run some tests
  - Component: BMI function definitions
  - Comparison: BMI vs non-BMI model output
- See more best practices: https://bmi.readthedocs.io/en/latest/bmi.best_practices.html

Further Development

BMI establishes an infrastructure extendable to further enhancements; e.g. serialization and calibration





Links to OWP Resources

Formulation	Language	NOAA-OWP Github
TOPMODEL	C	topmodel-bmi
CFE	C	cfe-bmi
Potential Evapotranspiration	C	pet-bmi
LSTM	Python	lstm-bmi
Noah-OWP-Modular	Fortran	noah-bmi
Soil-Freeze-Thaw	C++	soil-freeze-thaw-bmi

jessica.garrett@noaa.gov *water.noaa.gov*