

*Geoscientific Model Development Discussions* is the access reviewed  
discussion forum of *Geoscientific Model Development*

**NextFrAMES**

B. M. Fekete et al.

# Next generation framework for aquatic modeling of the Earth System

**B. M. Fekete<sup>1,\*</sup>, W. M. Wollheim<sup>2</sup>, D. Wisser<sup>2</sup>, and C. J. Vörösmarty<sup>1,\*</sup>**

<sup>1</sup>CUNY Environmental Cross-roads Initiative, The City College of New York at the City University of New York, 160 Convent Avenue, New York, New York, 10031, USA

<sup>2</sup>Complex Systems Research Center, Institute for the Study of Earth, Oceans and Space, University of New Hampshire, 39 College Rd., Durham, 03824, USA

\* also at: NOAA-Cooperative Remote Sensing Science and Technology Center, The City College of New York at the City University of New York, 160 Convent Avenue, New York, New York, 10031, USA

Received: 24 February 2009 – Accepted: 4 March 2009 – Published: 17 March 2009

Correspondence to: B. M. Fekete (bfekete@ccny.cuny.edu)

Published by Copernicus Publications on behalf of the European Geosciences Union.

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures

◀

▶

◀

▶

Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion



## Abstract

Earth System model development is becoming an increasingly complex task. As scientists attempt to represent the physical and bio-geochemical processes and various feedback mechanisms in unprecedented detail, the models themselves are becoming increasingly complex. At the same time, the complexity of the surrounding IT infrastructure is growing as well. Earth System models must manage a vast amount of data in heterogeneous computing environments. Numerous development efforts are on the way to ease that burden and offer model development platforms that reduce IT challenges and allow scientists to focus on their science. While these new modeling frameworks (e.g. FMS, ESMF, CCA, OpenMI) do provide solutions to many IT challenges (performing input/output, managing space and time, establishing model coupling, etc.), they are still considerably complex and often have steep learning curves.

The Next generation Framework for Aquatic Modeling of the Earth System (NextFrAMES, a revised version of FrAMES) have numerous similarities to those developed by other teams, but represents a novel model development paradigm. NextFrAMES is built around a modeling XML that lets modelers to express the overall model structure and provides an API for dynamically linked plugins to represent the processes. The model XML is executed by the NextFrAMES run-time engine that parses the model definition, loads the module plugins, performs the model I/O and executes the model calculations. NextFrAMES has a minimalistic view representing spatial domains and treats every domain (regardless of its layout such as grid, network tree, individual points, polygons, etc.) as vector of objects. NextFrAMES performs computations on multiple domains and interactions between different spatial domains are carried out through couplers. NextFrAMES allows processes to operate at different frequencies by providing rudimentary aggregation and disaggregation facilities.

NextFrAMES was designed primarily for hydrological modeling purposes, but many of its functionality should be applicable for a wide range of land surface models. In its present capabilities NextFrAMES is probably inadequate to implement fully coupled

## GMDD

2, 279–307, 2009

### NextFrAMES

B. M. Fekete et al.

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures



Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion



Earth System models, but future versions with the guidance from Earth System developers might someday eliminate its limitations. Our intent with NextFrAMES is to initiate a dialog about new ways of expressing models that is less tied to the actual implementation and allow scientist to develop models at a more abstract level.

## 1 Introduction

Earth System modeling is an increasingly challenging task due to the rapidly increasing complexity of the models themselves (Hall and O'Connell, 2007). Modern land surface, ecosystem, hydrological, etc. models represent biogeochemical and physical processes in such detail that model codes become very complicated. In addition to the complexity of the models themselves, the IT infrastructure upon which the models are built is growing more complex as well. While both computational power and mass storage media capacity have undergone exponential growth following Moore's law, the speed of accessing the mass storage devices clearly lagged behind. As a consequence, applications today are often limited by the read and write access of the vast amount of data required by the models. One way to reduce the limitations of slow storage devices is to trade data storage for processing on the fly. For instance, down-scaling algorithms built into models can reduce the amount of data that the models need to read. However, this solution comes at the price of performing computationally intensive interpolation algorithms as part of the model simulations. Such an approach essentially makes the already complicated models even more complex.

To complicate the matter further, the efficient management of the vast amount of data available to Earth Scientists requires sophisticated solutions. The era of "ad hoc" input/output using primitive, home-grown data formats is over; scientific data need a common management framework that provides catalogs, documentation, discovery mechanisms and preprocessing to feed the scientific applications. While emerging new data management methods and tools ultimately help the end-users, they put extra burden on the model developers to enable increasingly complex data services in

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures



Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion



already complex models.

When one considers that the computational power of single CPU core has remained more or less steady over the last four years while the Moore's law was realized by adding more cores to the CPUs (dual and quad core CPUs becoming common even in ordinary desktop computers) the need for implementing Earth System model components in a parallel processing environment is inevitable. Furthermore, parallel computers have vastly different architectures ranging from symmetric memory processing (SMP, when multiple processors have access to the same memory), non-uniform memory access (NUMA, when processors have access to faster memory individually and share common access to slower memory units) and cluster computers (e.g. Beowulf, when entire computers are clustered via fast network connections). These different platforms need very different parallelization strategies.

Computer science has delivered numerous essential components to make the utilization of these different architectures easier. For instance, Message Passing Interface (MPI) is commonly used in many parallel applications. No new development platform has emerged yet that could help to automate the parallelization of scientific tasks. When parallel computers became commonplace more than twenty years ago, one would have thought that new "non-procedural" languages (like LISP or Prolog) would emerge allowing the description of the computational tasks in a conceptual manner that intelligent interpreters or compilers could analyze and optimize to execute on parallel CPU systems. The last two or three decades brought numerous new programming languages into play (C++, Java, Python) and new concepts like structured and object oriented programming. The new languages primarily aided the construction of complex applications by compartmentalizing the application development and allowing higher levels of modularity, but these languages remained "procedural" in the sense that the code describes the tasks as a sequence without giving hints to the compiler or interpreter, what can be executed independently, what needs to precede other steps. Some languages incorporated complex event handling capabilities primarily to support the development of modern graphical user interfaces but the event handling concepts

[Title Page](#)

[Abstract](#)

[Introduction](#)

[Conclusions](#)

[References](#)

[Tables](#)

[Figures](#)



[Back](#)

[Close](#)

[Full Screen / Esc](#)

[Printer-friendly Version](#)

[Interactive Discussion](#)



are rarely suitable for Earth System models.

The lack of new development concepts might be due to the large differences in them demands of specific applications, so no common solution has yet emerged. As a consequence, we argue that Earth Scientists need to develop their own model development platform that is designed to address Earth Science problems. This new model development platform needs to be built on a new paradigm that allows the description of scientific tasks in a hardware platform independent manner.

## 1.1 Existing modeling frameworks

The need for unified modeling frameworks was realized long time ago and numerous solutions are emerging with different design goals. While all modeling frameworks are aimed at simplifying the integration of model components one way to categorize them might be the level at which the model integration occurs. The Flexible Modeling System<sup>1</sup> (FMS, developed at the Geophysical Fluid Dynamics Laboratory, GFDL) and the Earth System Modeling Framework<sup>2</sup> (ESMF, which might be viewed as descendant of FMS, was originally initiated by NASA, and currently under development at NCAR) both provide a series of utilities and services to model developers in the form of function libraries that provide solutions to various computational demands. ESMF offers function library tools for handling components (primarily grid) and associated arrays of variables, partitioning of components for execution on multiple processors, distribution of data to processors, exchange of data between processors and the gathering calculation results, management of time stepping, data exchanges between different components (coupling), input/output and log facilities. ESMF can be used as a primary mean for managing space and time in an Earth System model or it can be used as a wrapper around existing models that otherwise have their entirely different internal structure.

<sup>1</sup><http://www.gfdl.noaa.gov/~fms>

<sup>2</sup><http://www.esmf.ucar.edu/>

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures



Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion



While ESMF can be used as a “glue” between different models, its primary goal is to provide unified internal model infrastructure. In this respect, ESMF is a huge step forward as a new hardware-independent modeling platform, but the use of ESMF still has steep learning curve and requires the model developer to deal with low-level details (e.g. partitioning the model domains into computational subregions, managing the data distribution between compute nodes, etc.).

Open Modeling Interface<sup>3</sup> (OpenMI) funded by the European Commission and the Common Component Architecture<sup>4</sup> (CCA) initiated by the U.S. Department Energy provide solutions to higher level model coupling. Both OpenMI and CCA are designed to couple entire models (just like when ESMF is used in a “glue wrapper” mode. The capability to couple entire models is certainly useful, particularly the wrapping existing models into a larger modeling infrastructure, but at some point the legacy models will have to be retired as computer hardware and the software infrastructure makes them obsolete.

Perhaps the biggest shortcoming of coupling entire models is that as soon as these models maintain their own way of managing space and time (and consequently perform input/output in their own way), the necessary wrapper environment to handle the model differences needs to be fairly complicated. Furthermore, space and time management are exactly the most critical elements for parallel execution, so once they are embedded deeply in the model, any support to aid parallelization, and computational load balancing from the modeling framework standpoint becomes difficult if not impossible.

A third set of popular modeling environments came from commercial software vendors such as Matlab<sup>5</sup>, S-Plus<sup>6</sup> (and its open source equivalent R<sup>7</sup>), Stella, Simile<sup>8</sup>

<sup>3</sup><http://www.openmi.org>

<sup>4</sup><http://www.cca-forum.org>

<sup>5</sup><http://www.mathworks.com>

<sup>6</sup><http://www.insightful.com>

<sup>7</sup><http://www.r-project.org>

<sup>8</sup><http://www.simulistics.com>

[Title Page](#)[Abstract](#)[Introduction](#)[Conclusions](#)[References](#)[Tables](#)[Figures](#)[Back](#)[Close](#)[Full Screen / Esc](#)[Printer-friendly Version](#)[Interactive Discussion](#)

(Muetzelfeld and Masshender, 2003) and COMSOL<sup>9</sup>. These are probably the tools that are the closest to the new model development platforms that Earth system scientists will need. Actually, numerous Earth System model components have implementation using these tools (e.g. Topmodel written in Matlab, etc.). The common service that these tools provide is a simple means to perform computation with different levels of complexity on variable arrays, so that developers could focus on the actual computational tasks and let the software to perform rudimentary data management. While these software development platforms are often extremely valuable for quick prototyping, they don't scale well to the massive computational tasks of large Earth System models. Furthermore, these software tools are rarely geared to Earth science needs and due to the closed source nature of commercial software, adding the needed extensions is rarely feasible.

## 1.2 Common Earth System model function

The most important common functions that Earth System models and model components share is the discretization of space and time, linkages between different computational domains that discretize space and time differently (coupling) and calculations on the discrete spatial objects at some computational frequency. The discretization of time is perhaps more trivial since it is only a one dimensional problem and the major challenge is to map the regular time intervals to the somewhat irregular time increments in traditional calendars (e.g. varying length of month, leap years, the disconnect between weekly and monthly increments). A discretization of space is more challenging. Numerous solutions to numerically represent spatial features have emerged over the last few decades. The three characteristically different approaches are Geographical Information Systems, remote sensing image processing systems and Earth System Model tools. The primary focus of traditional GIS was to represent static spatial features via series of contours and polygons. Representing similar features in a gridded

<sup>9</sup><http://www.comsol.com>

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures



Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion



context was added later as a mean to make overlay calculations easier. The line between gridded GIS and image processing is blurry, but the biggest difference perhaps is that image processing tools need to manage not only static images, but the also changes that occur over time, yet typically at low temporal frequency (often in the order of daily to bi-weekly intervals). Earth system model tools (which don't really have a name on their own, but can be regarded as the tools supporting NetCDF<sup>10</sup> developed mostly by NCAR) were designed to manage space and time at high temporal frequencies by compromising on spatial resolution. These tools treat three or four dimensional arrays (representing 2D or 3D domains in time) as single objects.

The synergy of the GIS, image processing and Earth system tools is critical for the next generation of Earth System models to enable modelers to perform simulations over the computational domains that are most suitable for the particular model components and yet enable model components (operating on one type of spatial domain layout) to interface with other components. This synergy will require a generalized means of exchanging information between different domains (coupling).

### 1.3 Science expression in Earth System model components

Scientific representation of different Earth System model components often express the model structure in a form of block diagram (Fig. 1). The scientific thinking in these block diagrams have two components a) the block diagram layout and b) the processes in the boxes of the block diagram. An ideal modeling framework would allow scientists to focus on these two and alleviate the need to deal with core model functions such as performing model I/O, handling spatial objects and advancing time, or providing visualization.

This level of abstraction improves the model documentations and serves as the ultimate means to plug and play compatibility of model components. While the implementation of space and time management in such a modeling framework might be limiting

<sup>10</sup><http://www.unidata.ucar.edu/software/netcdf>

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures



Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion





for applications where novel approaches representing both is integral part of the science (e.g. adaptive finite difference meshes) as these technologies mature they could be implemented in the modeling frameworks.

## 2 Next generation framework for aquatic modeling of the Earth System

5 We have developed several modeling frameworks (Data, Assembler, FrAMES, and NextFrAMES) in the last 15 years following the previously described design goals. The latest effort building on the Framework for Aquatic Modeling of the Earth System (FrAMES) (Wollheim et al., 2008) was started about a year ago in response to the need identified in the ongoing NASA WaterNet Solution Networks and IDS projects.  
10 While FrAMES (as its name implies) was primarily designed to support hydrological and constituent transport and processing models, NextFrAMES eliminates most of its limitations and offer a modeling platform for wide range of land surface modeling applications.

15 NextFrAMES includes

- modeling XML (to describe the overall model structure),
- module plugin Application Programming Interface (API) for C/C++ (future versions of FrAMES will have bindings to other languages),
- I/O plugin API for different file and data services (OpenDAP, WaterML, etc.)
- 20 – run-time engine (currently designed for single CPU system, but we anticipate SMP capabilities next year implemented as part of the last year of our WaterNet effort)

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures



Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion



## 2.1 NextFrAMES model structure

NextFrAMES modeling XML (document type definition provided in Appendix A) organizes models around components (*aggregate*, *container*, *input*, and *region* (Fig. 2). Components represent spatial domains as a vector of computational objects. The actual handling of the physical domains and the translation from the domain elements (discrete points, grid cells, etc.) to the vector of objects is delegated to I/O plugin infrastructure. Components maintain lists of variables as a vectors (of the same length as the number of computation objects in the component). Components can contain any number of variables and parameters. Parameters are constant values through space and time.

While components themselves have no information about the actual layout (e.g. discrete points, grid, gridded networks) of its computational objects, it is the I/O plugin interface that provides the necessary object topologies. An object topology describes the spatial relationships between computational object. For instance, a 2D grid domain could define horizontal and vertical neighbors (2D topology). Gridded river networks (Fekete et al., 2001; Vörösmarty et al., 2000) could define downstream and upstream river reach (tree topology). Components could have any number of topologies and the available topologies determine the available spatial operators (derivative, route) the can be requested. Each component has its own temporal frequency at which the component operates.

Components can form a hierarchical tree through *container* components. Container components hold other components, *interfaces* and processing *modules*. Container components inherit their vector of computational objects form other components. Currently only *input* components can define new vector of computational object.

Aggregate components (Example 2) provide a special means to perform aggregate functions (mean, minimum, maximum, standard deviation, percentage) over a *variable* of source *component* and map the result to the vector of computational objects in the *aggregate* component. For instance, *aggregate* components can be used to

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures



Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion



downscale high resolution land-use/cover information into distribution of cover types (as percentages) at a coarser resolution, or calculate aggregate statistics (mean, minimum, maximum) from fine resolution input to a coarser resolution domain.

Container components (Example 3) are the most important elements of the model definition. *Containers* provide the means to exchange variables from one component to another through *interfaces* and hold the *modules* to perform calculation tasks. An *interface* maps a *variable* from a source component to a new *variable* in the *container* component via coupler. The coupler is a simple weighting mechanism (established by the run-time engine) assigning a set of object from the source component and the corresponding weighting to a particular object in the destination component (Fig. 2). Couplers are one directional, but separate couplers from components can be established in each direction. Establishing couplers is hidden to the modeler and it is NextFrAMES' duty to find out how to couple two different set of computational objects.

The main purpose of *container* components to perform computations on the *variables* within the container via process modules provided in dynamically linked plugin (Example 4). A *process* module plugin (that is written in high level programming language, currently only in C) has two functions (initialize, execute). The *initialize function* is called in the parsing phase of the model execution. The role of the *initialize function* is to return the list of parameters and input and output variables of the module to NextFrAMES. The *process* modules can define default value and plausible value range<sup>11</sup> for each requested parameter. Variables can have default value (to overwrite when missing values occur) and unit attributes. The execute function is called in the model calculation phase for each computational object in the *container* component repeatedly for each time step. The NextFrAMES takes care of converting variables to the requested units and place the *parameters* and *variables* into a single user defined

<sup>11</sup>The plausible value range for parameters is not only allowing the run-time environment to warn the user that the user defined parameter values is outside of the intended range according to the module's author but one could envision higher level run-time engines developed in the future that could automate the optimization of parameters using Monte-Carlo type simulations.

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures



Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion



record. The execute functions responsibility is to read the passed variable values and return the results in the output variable elements of the user defined record.

The *variables* and *parameters* of a *process* module are associated to *variables* and *parameters* defined in the container components via *aliases*, therefore the same *process* module can be reused within the same models for different *variables* and *parameters*.

Simplified mechanism to define modules is provided through equations allowing the symbolic definition of calculation within the modeling XML (Example 5). The *equation* module uses *alias* mechanisms to relate symbolic variables to *variables* and *parameters* of the *container* component.

*Parameters* (that are uniform in space and time during model calculations) can be defined at any level of the components hierarchy down to the module level, which determine the scope of the *parameters*. For instance, *parameters* defined in the “root” container (which is the model clause of the modeling XML) have global scope (i.e. known throughout the whole model). *Parameters* defined in descendent *container* component are only known within the *container* and in any of it descendent *containers*. *Parameters* defined in modules are only known in the module where they were defined.

Derivative and route operator modules (Example 6) perform calculations across topologically linked computational objects. Derivatives calculate forward, backward and center differences in one of the axis direction on 2D or 3D topology. Route modules perform upstream accumulation on objects with tree topology (river network).

Input variables in any module (*process*, *equation*, *derivative* or *route*) have to be defined before the module is specified. Normally, variables are either defined through *interfaces* or as outputs from previous modules. Variables that are updated in subsequent module calculations can be defined as *initial* variable. Uniform initial value of the initial variables can be provided in the model XML or spatially varying values from previous model runs can be loaded through “restart” or “state” files defined as container attributes.

[Title Page](#)

[Abstract](#)

[Introduction](#)

[Conclusions](#)

[References](#)

[Tables](#)

[Figures](#)



[Back](#)

[Close](#)

[Full Screen / Esc](#)

[Printer-friendly Version](#)

[Interactive Discussion](#)



The purpose of input components (Example 7) is to translate variables from input sources the vector of computational objects (Fig. 2). An input component uses an I/O plugin to access the content of the input source and retrieve optional object topologies. The I/O plugin is specified in the modeling XML along with the source URL of the external data. Input components can inherit the time step from the input source or specify the input frequency via the “time\_step” attribute. When the requested frequency is slower than the time step of the input data, the input component performs temporal averaging for the requested frequency. When the input data has lower frequency than the requested time step, the same values are returned at each high frequency data request as a step function. The “offset” attribute (not shown in example) can be used to request low frequency input records in advance or delayed. The offset capabilities will allow the implementation of advanced downscaling (beyond the simple step function) such as bilinear, spline, etc interpolation.

Region components provide a simple means to subset the vector of computational objects, based on simple criteria on a single variable from the source components. For instance, one could implement a plant growth model in NextFrAMES and define bare soil regions (that would vary over time) as a function of vegetation state. Regions actually inherit the layout of the parent component but allow masking inactive objects.

*Container* components could include outputs requests to write out any variables as the model simulation progresses (Example 8). The output request uses the I/O plugin of the component from which the container inherited its layout (which is always an input component since only input component can define layout).

## 2.2 NextFrAMES run-time engine

The NextFrAMES run-time engine executes the model components in the order as they are defined in the modeling XML (Fig. 3). *Processes* defined in *container* component are also executed in the order as they are defined. The first reference implementation of the NextFrAMES run-time engine will be completed in the first quarter of 2009 as part of CCNY contribution to NASA-WaterNet. The reference implementation is designed

[Title Page](#)[Abstract](#)[Introduction](#)[Conclusions](#)[References](#)[Tables](#)[Figures](#)[Back](#)[Close](#)[Full Screen / Esc](#)[Printer-friendly Version](#)[Interactive Discussion](#)

for single processor systems. SMP capabilities are anticipated by the end of the 2009. Over time, we would hope to see multiple implementation of the run-time engine to emerge just like there are multiple implementations for web-browsers (Mozilla, Opera, MS Internet Explorer). The different implementation can be optimized for different platforms (ranging from single CPU systems to cluster computers). For instance, an implementation of the NextFrAMES run-time engine for cluster computers could be built on top of ESMF, which already has numerous components needed to manage components with different layouts and couple different domains.

While NextFrAMES is not likely to suit every Earth system model needs, but it certainly can provide a strong basis for a wide range of applications. At the minimum, NextFrAMES is an ultimate “Swiss Army Knife” tool for manipulating time varying spatial data products. The ability of interfacing between components at different spatial and temporal frequencies makes NextFrAMES a unique data processing tool that can perform spatial and temporal up and downscaling while performing addition complex calculations. The NextFrAMES modeling XML along with the plugin API has the potential to change the model development paradigm similarly to how HTML changed dramatically the sharing and distributing textual content.

### 3 Conclusions

The main difference between NextFrAMES and existing modeling frameworks is that NextFrAMES is intended to provide a high level abstraction of the scientific tasks in model development and hides largely the common functionality that are targeted by other frameworks.

The NextFrAMES modeling XML and plugin infrastructure provides flexible basis to develop wide range of Earth System model components (land surface, ecosystem and hydrological models). Its operational predecessor (FrAMES) demonstrated the feasibility of the new modeling approach and provided flexible development platform. Deficiencies in FrAMES are largely addressed in the NextFrAMES, design ,which will be

[Title Page](#)

[Abstract](#)

[Introduction](#)

[Conclusions](#)

[References](#)

[Tables](#)

[Figures](#)



[Back](#)

[Close](#)

[Full Screen / Esc](#)

[Printer-friendly Version](#)

[Interactive Discussion](#)



released to the scientific community simultaneously with the publication of the present paper.

The first implementation of the run-time engine that parses and interprets the modeling XML loads the module and input/output plugins and performs the model simulation is designed for single CPU system, but a parallel version working on shared memory systems is expected by the end of 2009. The parallel and the single CPU implementations of the run-time engine will be able to execute the same modeling XML and corresponding plugins without on single or multi processor machines without modifications. While the current set of input/output plugins are limited to NetCDF data format, additional input/output plugins can be written without affecting the model implementation.

The NextFrAMES run-time engines developed at The City College of New York will be released as open source project through the popular SourceForge archive. The intention is to turn NextFrAMES a freely available model development platform that would attract not only model developers to implement their models within the NextFrAMES infrastructure, but computer scientists to add new features such as data handling capabilities to other file formats beyond NetCDF by developing additional input/output plugins or different implementation of the run-time engine optimized from various platforms.

The NextFrAMES modeling platform will provide new opportunities to develop support infrastructure. One can envision graphical model builder graphical interface (similar to the one in Stella, Simile or Ccaffeine GUI of the Common Component Architecture). Additional tools could provide new means to link the models to metadata and data services so the modeler could choose from available input data meeting the input requirements of the model simulations and access the data through data services (Horak et al., 2008) (e.g. OpenDAP and OGC WFS data). Surrounding support infrastructure could perform caching input data (similarly to web-browsers caching large web contents) such that subsequent execution of the same model with slight change in configuration could take advantage of the data retrieved during previous model runs.

[Title Page](#)

[Abstract](#)

[Introduction](#)

[Conclusions](#)

[References](#)

[Tables](#)

[Figures](#)



[Back](#)

[Close](#)

[Full Screen / Esc](#)

[Printer-friendly Version](#)

[Interactive Discussion](#)



Some of the support tools are already under development at The City College of New York supporting and at the University of New Hampshire for both the Global Terrestrial Network for Hydrology (GTN-H, <sup>12</sup>) effort of the World Meteorological Organization and the end-to-end water resources management tools developed as CCNY/UNH's contribution the the State-of-the Global Water Systems initiative of the Global Water Systems Project.

## Appendix A

```
!-- DTD for Land surface model definition file -->

<!ELEMENT model      (aggregate|container|input|parameter|region)+>
<!ATTLIST model      name CDATA #REQUIRED
                    version CDATA #REQUIRED
                    begin CDATA #REQUIRED
                    end CDATA #REQUIRED>

<!ELEMENT aggregate  (category|mean|minimum|maximum|stdev)+>
<!ATTLIST aggregate name CDATA #REQUIRED
                    layout CDATA #REQUIRED
                    component CDATA #REQUIRED
                    variable CDATA #REQUIRED>

<!ELEMENT container  (aggregate|container|derivative|equation|initial|
interface|output|parameter|probe|process|region|route)+>
<!ATTLIST container name CDATA #REQUIRED
                    layout CDATA #REQUIRED
                    time_step CDATA #REQUIRED
                    states CDATA #REQUIRED>

<!ELEMENT equation   (alias|parameter)*>
<!ATTLIST equation  name CDATA #REQUIRED
                    equation CDATA #REQUIRED
                    unit CDATA #IMPLIED>

<!ELEMENT input      (variable)+>
<!ATTLIST input      name CDATA #REQUIRED
                    time_step CDATA #REQUIRED
                    url CDATA #REQUIRED
                    plugin CDATA #REQUIRED
                    method CDATA #REQUIRED>

<!ELEMENT output     (variable)+>
```

<sup>12</sup><http://www.gtn-h.net>

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures

◀

▶

◀

▶

Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion





```

<!ATTLIST output      name CDATA #REQUIRED
                        path CDATA #REQUIRED>

<!ELEMENT process     (alias|parameter)*>
<!ATTLIST process     name CDATA #REQUIRED
                        plugin CDATA #REQUIRED
                        method CDATA #REQUIRED>

<!ELEMENT region      (alias)>
<!ATTLIST region      name CDATA #REQUIRED
                        condition CDATA #REQUIRED>

<!ELEMENT extent      (#EMPTY)>
<!ATTLIST extent      direction (x|y|z) #REQUIRED
                        minimum CDATA #REQUIRED
                        maximum CDATA #REQUIRED>

<!ELEMENT dimension   (#EMPTY)>
<!ATTLIST dimension   direction (x|y|z) #REQUIRED
                        size CDATA #REQUIRED>

<!ELEMENT point       (#EMPTY)>
<!ATTLIST point       name CDATA #REQUIRED
                        xcoord CDATA #REQUIRED
                        ycoord CDATA #REQUIRED>

<!ELEMENT interface   (#EMPTY)>
<!ATTLIST interface   name CDATA #REQUIRED
                        relation (root|parent|own) #REQUIRED
                        component CDATA #REQUIRED
                        variable CDATA #REQUIRED
                        default CDATA #REQUIRED>

<!ELEMENT alias       (#EMPTY)>
<!ATTLIST alias       name CDATA #REQUIRED
                        alias CDATA #REQUIRED
                        type (variable|parameter) #IMPLIED>

<!ELEMENT category    (#EMPTY)>
<!ATTLIST category    name CDATA #REQUIRED
                        values CDATA #REQUIRED>

<!ELEMENT derivative  (#EMPTY)>
<!ATTLIST derivative  name CDATA #REQUIRED
                        variable CDATA #REQUIRED
                        direction (x|y|z) #REQUIRED
                        difference (centered|backward|forward) #IMPLIED>

<!ELEMENT initial     (#EMPTY)>
<!ATTLIST initial     name CDATA #REQUIRED
                        unit CDATA #REQUIRED
                        initial_value CDATA #IMPLIED>

<!ELEMENT parameter   (#EMPTY)>
<!ATTLIST parameter   name CDATA #REQUIRED
                        value CDATA #REQUIRED >

```

# GMDD

2, 279–307, 2009

## NextFrAMES

B. M. Fekete et al.

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures

◀

▶

◀

▶

Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion



```

<!ELEMENT probe          (#EMPTY)>
<!ATTLIST probe
  name CDATA #REQUIRED
  path CDATA #REQUIRED
  xcoord CDATA #REQUIRED
  ycoord CDATA #REQUIRED>

<!ELEMENT route          (#EMPTY)>
<!ATTLIST route
  name CDATA #REQUIRED
  unit CDATA #REQUIRED>

<!ELEMENT variable       (#EMPTY)>
<!ATTLIST variable
  name CDATA #REQUIRED
  standard_name CDATA #REQUIRED>

```

## References

- Fekete, B. M., Vörösmarty, C. J., and Lammers, R. B.: Scaling gridded river networks for macro-scale hydrology: Development and analysis and control of error, *Water Resour. Res.*, 37, 1955–1968, 2001. 288
- Hall, J. and O’Connell, E.: Earth systems engineering: turning vision into action, *Proceedings of ICE, Civil Engineering* 160, 114–122, 2007. 281
- Horak, J., Orlik, A., and Stromsky, J.: Web services for distributed and interoperable hydro-information systems, *Hydrol. Earth Syst. Sci.*, 12, 635–644, 2008, <http://www.hydrol-earth-syst-sci.net/12/635/2008/>. 293
- Muetzelfeld, R. and Masshender, J.: The Simile visual modelling environment, *Eur. J. Agron.*, 18, 345–358, 2003. 285
- Vörösmarty, C. J., Fekete, B. M., Meybeck, M., and Lammers, R. B.: Global System of Rivers: Its role in organizing continental land mass and defining land-to-ocean linkages, *Global Biochem. Cy.*, 14, 599–621, 2000. 288
- Wollheim, W. M., Vörösmarty, C. J., Bouwman, A. F., Green, P. A., Harrison, J., Linder, E., Peterson, B. J., Seitzinger, S., and Syvitski, J. P. M.: Global N removal by freshwater aquatic systems: a spatially distributed, within-basin approach, *Global Biogeochem. Cy.*, GB2026, doi:10.1029/2007GB002963, 2008b. 287

**GMDD**

2, 279–307, 2009

**NextFrAMES**

B. M. Fekete et al.

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures

◀

▶

◀

▶

Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion



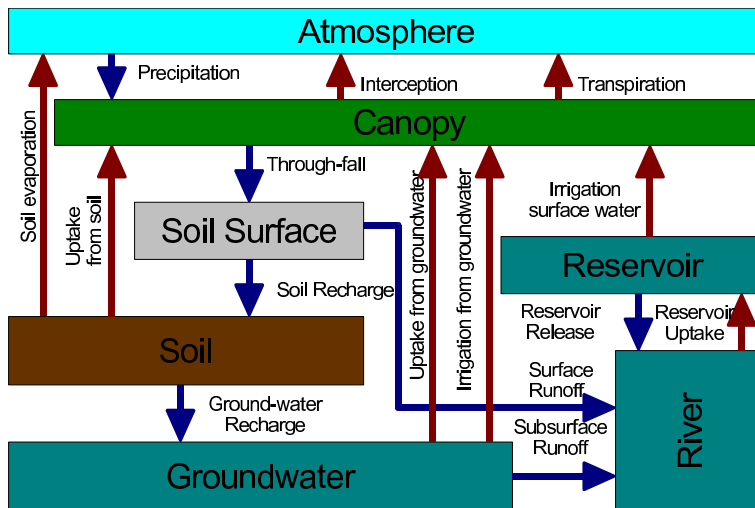


Fig. 1. An example of a conceptual Hydrological Model following the paths of water in the water cycle.

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures

◀

▶

◀

▶

Back

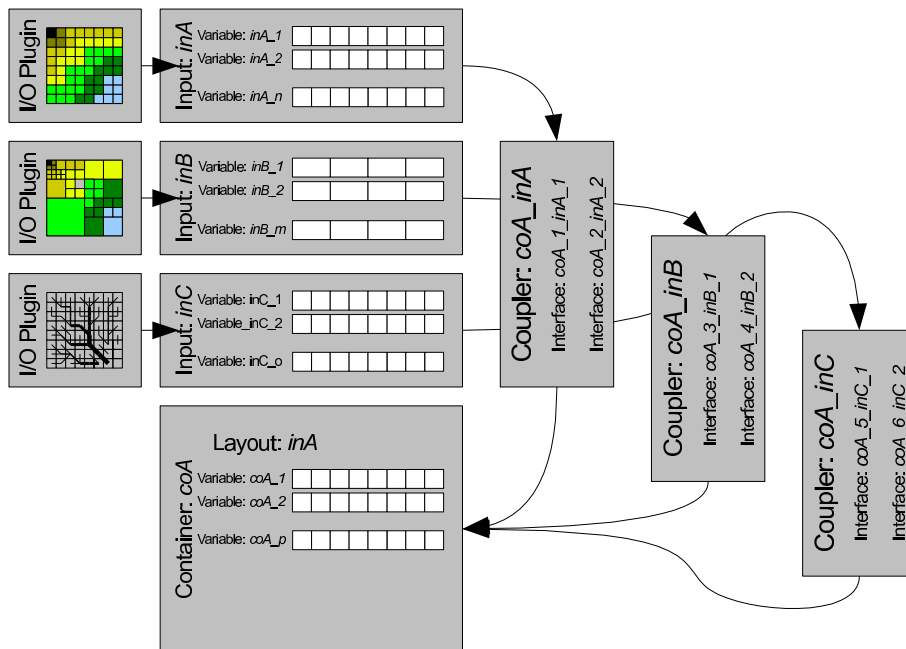
Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion





**Fig. 2.** NextFrAMES Components layout. Input components provide mapping from spatial domains (grid, polygons, triangular mesh, gridded network, etc.) to the variable vectors in the run-time engine. Couplers establish the mappings between the vectors of object from one component to objects from another component. Interfaces associated with couplers transfer the variable values from the component to component.

[Title Page](#)
[Abstract](#)
[Introduction](#)
[Conclusions](#)
[References](#)
[Tables](#)
[Figures](#)
[Back](#)
[Close](#)
[Full Screen / Esc](#)
[Printer-friendly Version](#)
[Interactive Discussion](#)


```

<!DOCTYPE lsm SYSTEM "lsm.dtd">
<!-- Multi layout Hydrological Model -->
<model name="Multi-layout Hydrological Model" version="1.0" begin="2000-01-01" end="2007-12-31">
  <input name="air temperature" time_step="1 day" url="file://airtemp.nc" plugin="NCGridRegular.iplugin" method="NCGridRegular">
    <!-- air temperature input -->
    <variable name="air temperature" standard_name="airtmp"/>
  </input>
  <input name="precipitation_input" time_step="inherit" url="file://precip.nc" plugin="NCGridRegular.iplugin" method="NCGridRegular">
    <!-- Precipitation input -->
    <variable name="precipitation" standard_name="precip"/>
  </input>
  <input name="reanalysis_input" time_step="inherit" url="file://ncep.nc" plugin="NCGrid.iplugin" method="NCGrid">
    <!-- NCEP Reanalysis input -->
    <variable name="vapor pressure" standard_name="vapor_pressure"/>
    <variable name="wind speed" standard_name="wind_speed"/>
  </input>
  <input name="landcover_input" time_step="inherit" url="file://landcover.nc" plugin="NCGrid.iplugin" method="NCGrid">
    <!-- Landcover input -->
    <variable name="land_cover" standard_name="land_cover"/>
  </input>
  <aggregate name="landcover_percent" layout="precipitation_input" component="landcover_input" variable="landcover">
    <!-- Characteristic land cover types -->
    <category name="deciduous forest" values="1,5,7"/>
    <category name="conifer forest" values="10-12"/>
    <category name="evergreen forest" values="8"/>
    <category name="tall grass" values="9"/>
    <category name="short grass" values="18"/>
    <category name="savannah" values="14,16"/>
    <category name="shrubland" values="17"/>
    <category name="tundra" values="19"/>
    <category name="water" values="20"/>
    <category name="other" values="unused"/>
  </aggregate>
  <input name="land_type" time_step="inherit" url="file://landtype.nc" plugin="NCGridRegular.iplugin" method="NCGridRegular">
    <!-- Land alterations -->
    <variable name="natural" standard_name="natural"/>
    <variable name="wetland" standard_name="wetland"/>
    <variable name="cultivated" standard_name="cultivated"/>
    <variable name="impervious" standard_name="impervious"/>
  </input>
</model>

```

Fig. 3. NextFrAMES Model XML in Eclipse XML editor.

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures

◀

▶

◀

▶

Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion



```
<output name="water budget" path="water_budget.nc">
<variable name="wetland" standard_name="wetland" />
</output>
```

**Example 1.** Output request in container components export any variable to the output file specified by the “path” attribute. The output request is carried out using the I/O plugin of the root component of the container component from which it inherited its layout.

Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures



Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion



```

<aggregate name="landcover_percent"
layout="precipitation_input"
component="landcover_input" variable="landcover">
<!--Characteristicland cover types-->
<category name="decidious forest" values="1,5,7"/>
<category name="conifer forest" values="10-12"/>
<category name="evergreen forest" values="8"/>
<category name="tall grass" values="9"/>
<category name="short grass" values="18"/>
<category name="savannah" values="14,16"/>
<category name="shrubland" values="17"/>
<category name="tundra" values="19"/>
<category name="water" values="20"/>
<category name="other" values="unused"/>
</aggregate>

```

**Example 2.** The aggregate example referred as “landcover\_precent” inherits its layout (vector of computational objects) from the component named (“precipitation\_input”) and aggregates the “landcover” variable of the “landcover\_input” component into variables called “decidious forest”, “conifer forest”, “evergreen forest”, etc. by calculating the percentage of occurrences of categorical values (listed in the values attributes).

[Title Page](#)
[Abstract](#)
[Introduction](#)
[Conclusions](#)
[References](#)
[Tables](#)
[Figures](#)
[◀](#)
[▶](#)
[◀](#)
[▶](#)
[Back](#)
[Close](#)
[Full Screen / Esc](#)
[Printer-friendly Version](#)
[Interactive Discussion](#)


```
<container name="land-surface" layout="land_type"
time_step="1 day" states="landsurf_state.nc">
<interface name="air_temperature"
component="air_temperature" relation="parent"
variable="air_temperature" default="nodata"/>
<interface name="precipitation"
component="precipitation_input" relation="parent"
variable="precipitation" default="nodata"/>
<interface name="natural" component="land_type"
relation="sibling" variable="natural"
default="0.0"/>
</container>
```

**Example 3.** Container component example inherits its layout from a component called “land\_type” defined in the modeling XML before the container component named “land-surface”. The container component can define its own temporal frequency through “time\_step” attribute and provide optional input source for initial variables through the “states” attribute.

[Title Page](#)[Abstract](#)[Introduction](#)[Conclusions](#)[References](#)[Tables](#)[Figures](#)[◀](#)[▶](#)[◀](#)[▶](#)[Back](#)[Close](#)[Full Screen / Esc](#)[Printer-friendly Version](#)[Interactive Discussion](#)



```
<process name="potential evapotranspiration"  
plugin="wbn.mplugin" method="Hamon">  
<alias name="air temperature" alias="airtemp"  
type="variable"/>  
<alias name="vapor pressure" alias="vpress"  
type="variable"/>  
<parameter name="coefficient" value="25.0"/>  
</process>
```

**Example 4.** Process modules in container components define the source of dynamically loaded plugin file through plugin attribute and the actual module functions within the plugin via the method attributes. Variables and parameters used in the plugin function are mapped to variables in the container component (encapsulating the process module) via alias mechanism.

[Title Page](#)[Abstract](#)[Introduction](#)[Conclusions](#)[References](#)[Tables](#)[Figures](#)[⏪](#)[⏩](#)[◀](#)[▶](#)[Back](#)[Close](#)[Full Screen / Esc](#)[Printer-friendly Version](#)[Interactive Discussion](#)

```
<equation name="akarmi" unit="m^3/s"  
equation="([a] + [b]) * [d]" >  
<alias name="precip" alias="a" type="variable"/>  
<alias name="airtemp" alias="b" type="variable"/>  
<alias name="coeff" alias="d" type="parameter"/>  
</equation>
```

**Example 5.** Equations provide simplified mechanism to define modules. The result of the equations is returned in variable named as the name attribute of the equation. The symbols in the equation are mapped to variables in the container component via aliases.

[Title Page](#)[Abstract](#)[Introduction](#)[Conclusions](#)[References](#)[Tables](#)[Figures](#)[◀](#)[▶](#)[◀](#)[▶](#)[Back](#)[Close](#)[Full Screen / Esc](#)[Printer-friendly Version](#)[Interactive Discussion](#)

```
<derivative name="water_table_dx" direction="x"  
difference="centered" variable="groundwater" />  
<route name="discharge" unit="m^3/s"  
variable="runoff" />
```

**Example 6.** Derivative and route operators perform calculations across multiple object related to each other through object topology. Derivative operators require 2D or 3D topology to calculate centered, forward or backward difference in x, y or z direction. Route operators sum the values of the specified variable upstream and propagate the accumulated sum downstream.

[Title Page](#)[Abstract](#)[Introduction](#)[Conclusions](#)[References](#)[Tables](#)[Figures](#)[◀](#)[▶](#)[◀](#)[▶](#)[Back](#)[Close](#)[Full Screen / Esc](#)[Printer-friendly Version](#)[Interactive Discussion](#)

```
<input name="precipitation_input"  
time_step="inherit" url="file://precip.nc"  
plugin="NCgridRegular.iplugin" method="NCgridRegular">  
<variable name="precipitation" standard_name="precip" />  
</input>
```

**Example 7.** Input component named “precipitation\_input” provide the mean to read variables as boundary condition from input source (specified through the “url” attribute). The actual input/output is performed via plugin methods that are dynamically loaded at run time.

[Title Page](#)

[Abstract](#)

[Introduction](#)

[Conclusions](#)

[References](#)

[Tables](#)

[Figures](#)

[◀](#)

[▶](#)

[◀](#)

[▶](#)

[Back](#)

[Close](#)

[Full Screen / Esc](#)

[Printer-friendly Version](#)

[Interactive Discussion](#)



Title Page

Abstract

Introduction

Conclusions

References

Tables

Figures



Back

Close

Full Screen / Esc

Printer-friendly Version

Interactive Discussion

```
<output name="water budget" path="water_budget.nc">  
<variable name="wetland" standard_name="wetland"/>  
</output>
```

**Example 8.** Output request in container components export any variable to the output file specified by the “path” attribute. The output request is carried out using the I/O plugin of the root component of the container component from which it inherited its layout.

