CSDMS

COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

SEMI-ANNUAL REPORT

JULY 30, 2011

NSF COOPERATIVE AGREEMENT 0621695

CSDMS

COMMUNITY SURFACE DYNAMICS MODELING SYSTEM

Preface

CSDMS is the virtual home for a diverse community of experts who foster and promote the modeling of earth surface processes, with emphasis on the movement of fluids, sediment and solutes through landscapes, seascapes and through their sedimentary basins. CSDMS develops, integrates, disseminates & archives software that reflects and predicts earth surface processes over a broad range of time and space scales. CSDMS deals with the Earth's surface—the ever-changing, dynamic interface between lithosphere, hydrosphere, cryosphere, and atmosphere. CSDMS employs state-of-the-art architectures, interface standards and frameworks that make it possible to convert stand-alone models into flexible, "plug-and-play" components that can be assembled into larger applications. The CSDMS model-coupling environment offers language interoperability, structured and unstructured grids, and serves as a migration pathway for surface dynamics modelers towards High-Performance Computing (HPC). This Semi-Annual Report covers the period from March 2011 to July 2011, and provides an update since the last 2010 Annual Report to NSF.

Table of Contents

Just the Facts 4

Progress on Year 5 Goals (April – July, 2011) 11

Appendix 1: A Component-Based Approach to Integrated Modeling in the Geosciences: The Design of CSDMS 29

CSDMS 'JUST THE FACTS'

CSDMS MODEL REPOSITORY

The CSDMS **Model Repository** offers metadata and links to 180 CSDMS-related models: 72% are available for download through the CSDMS web site (e.g. CHILD, SedFlux); 28% available after separately registering with other community efforts (e.g. ROMS, NearCOM). Models include landscape/seascape evolution models, morphodynamics models, transport models, climate and ocean models, and comprising 3.5 million lines of code written in ten languages.

Repository statistics as of July 2011: csdms.colorado.edu/wiki/Model_SLOC_Page

Language	Projects	Comment	Source	Total
Fortran 77/90/95+	37	627882	1420763	2048645
c/c++	63	270959	954826	1225785
Python	6	43221	109943	153164
MATLAB	13	14766	31310	46076
IDL	1	16730	18426	35156
Statistical Analysis Software	1	2390	5796	8186
Java	1	1107	6422	7529
Visual Basic	1	537	5735	6272
Total	123	977592	2553221	3530813

Models and Modeling Tools by Environmental Domain csdms.colorado.edu/wiki/Model_download_protal

113	Terrestrial
43	Coastal
30	Marine
80	Hydrology
6	Climate
2	Carbonate

Model code is downloaded in aid of science discovery ~2500 times per year. Models can be run on the CSDMS supercomputer without download and are not included in these statistics. Community models downloaded from other sites (e.g. ROMS, NearCOM) are also not counted. The top ten most downloaded models by version (July 2011):

	Model	No. Times	Topic
1.	**topotoolbox**	*823*	A set of Matlab functions for topographic analysis
2.	**topoflow**	*613*	Spatially-distributed, D8-based hydrologic model
3.	**child**	*612*	Landscape evolution model
4.	**sedflux**	*251*	Basin filling stratigraphic model
5.	**hydrotrend**	201	Climate driven hydrological transport model
6.	**2dflowvel**	*189*	Tidal & wind-driven coastal circulation routine
7.	**adi-2d**	*184*	Advection Diffusion Implicit method for 2D diffusion
8.	**bing**	*169*	Submarine debris flows
9.	**midas**	*158*	Coupled flow- heterogeneous sediment routing model
10.	**gc2d**	*135*	Glacier / ice sheet evolution model

CSDMS DATA REPOSITORY csdms.colorado.edu/wiki/Data_download

Data Repository as of July, 2011

Data Type	Databases
Topography/bathy	16
Climate	5
Hydrography	5
River discharge	8
Cryosphere	5
Soils	2
Land cover	2
Substrates	3
Human Dimensions	2
Sea level	1
Oceanography	9
GIS Tools	12
Network Extraction	8

CSDMS EDUCATION REPOSITORY

The **Education Repository** offers undergraduate and graduate modeling courses, educational modules, modeling labs, and process and simulation movies.

Animations library csdms.colorado.edu/wiki/Movies_portal.

Climate & Oceanographic Animations	8
Terrestrial Animations	16
Coastal Animations	21
Marine Animations	9
Laboratory Movies	14
Real Event Movies	31

Image Library csdms.colorado.edu/wiki/Images_portal

Terrestrial Images	90
Coastal and Marine Images	49

Modeling Labs csdms.colorado.edu/wiki/Labs_portal

1. Glacio-Hydrological Modeling
2. Modeling River-Delta Interactions
3. Sediment Supply Numerical Experiments
4. Landscape Evolution Numerical Experiments
5. Earth Science Models for K6-12
6. Hydrological Processes Exercises
7. Sinking Deltas
8. Coastal Stratigraphy Numerical Experiments

Modeling Lectures and Courses csdms.colorado.edu/wiki/Lectures_portal

1. Surface Dynamics Modeling with CMT — I Overeem & SD Peckham
2. Quantitative Earth-surface Dynamics Modeling — JPM Syvitski
3. 1D Sediment Transport — G Parker
4. Morphodynamics of Rivers — G Parker
5. Source to Sink Systems around the World — Keynote Chapman Lectures
6. Plug and Play Component Technology — JPM Syvitski
7. Geological Modeling — I Overeem

Modeling Textbooks csdms.colorado.edu/wiki/Modeling_Textbooks

1. Quantitative Modeling of Earth Surface Processes *By: Pelletier, J.D.*
2. Simulating Clastic Sedimentary Basins: Physical Fundamentals and Computing Procedures *By: R.L. Slingerland, K. Furlong and J. Harbaugh*
3. 1D Sediment Transport Morphodynamics with applications to Rivers and Turbidity Currents *By: G Parker*

CSDMS EXPERIMENTAL SUPERCOMPUTER csdms.colorado.edu/wiki/HPCC_information

The CSDMS High Performance Computing Cluster has operational issues during the Spring period causing periodic shutdowns. These issues have been addressed by SGI. Over 130 CSDMS members now have accounts on the system and have met the use criteria:

- Running a CSDMS model(s) to advance science
- Developing a model that will ultimately become part of the CSDMS model repository.
- Developing a new data systems or visualizations in support of CSDMS models.

CSDMS High Performance Computing Cluster (HPCC) System *Beach* is an SGI Altix XE 1300 with 88 Altix XE320 compute nodes (704 cores, 3.0 GHz E5472 Harpertown processors) (≈8 Tflops). 64 nodes have 2 GB of memory per core, 16 nodes have 4 GB of memory per core. *Beach* is controlled through an Altix XE250 head node. Internode communication is accomplished through a non-blocking InfiniBand fabric. Each compute node has 250 GB of local temporary storage. All nodes can access 72TB of RAID storage through NFS. *Beach* provides GNU and Intel compilers as well as their MPI counterparts (mvapich2, mpich2, and openmpi). The main power management is an APC UPS with 30 minutes of uptime at 50% load. *Beach* head-nodes are backed-up by a separate SGI installed UPS system. *Beach* is supported by the CU ITS Managed Services (UnixOps) under contract to CSDMS. Hardware upgrades (nodes, memory, storage) is scheduled for the later part of 2011.

Beach will soon be directly linked to the *Janus* supercomputer, funded in part by NSF under Grant No. CNS-0821794. The Janus system consists of 1368 nodes, each containing two 2.8 GHz Intel Westmere processors with six cores each (16,416 cores total) and 24 GB of memory (2 GB/core). The nodes are connected using a fully non-blocking quad-data rate InfiniBand interconnect, and the system's initial deployment will provide about 1 PB of parallel temporary disk storage. This system will be available to CU-Boulder researchers and collaborators. Additionally, CRC provides of a small "Analytics and Visualization" cluster where each node will has 48 cores and 0.5 TB of memory for data intensive applications and pre- and post-processing.

Projects that significantly use the HPCC http://csdms.colorado.edu/wiki/HPCC_projects

Some CSDMS member's scientific projects heavily rely on the CSDMS High Performance Computing Cluster, e.g. :

1. Coupling fluvial discharge and coastal evolution models (*Ashton, Kettner, Xing*)
2. Hydrodynamics and Sediment-Transport in the Poverty Bay Portion of the Waipaoa Sedimentary System (*Harris, McNinch*)
3. Investigating valley spacing regularity on evolving mountain fronts (*Capolongo, Refice, Lovergine, Ranaldo*)
4. Lithology Image Strips Extraction for the Ocean Drilling Program (*Jenkins*)
5. Niger Delta Project (*Hannon, Kettner, Syvitski, Peckham*)
6. Numerical Modeling of Permafrost Dynamics in Alaska using a High Spatial Resolution Dataset (*Marchenko, Jafarov*)
7. Numerical simulations of turbidity and gravity currents interacting with complex topographies (*Nasr-Azadani, Radhakrishnan*)
8. Repeat glacier elevation and velocity maps from multi-view stereophotography (*Welty*)
9. Surface Process Modeling Using CMT Course (Instructors: *Overeem, Peckham*)
10. The BQARTwbm distributed sediment flux model (*Cohen, Kettner, Syvitski, Fekete*)

The impact of thermocline induction on decadal variability of the North Atlantic carbon sink (*Lovenduski*)

CSDMS WEB PORTAL STATISTICS csdms.colorado.edu/wiki/Special:Statistics

Content Pages	971	Page Edits	32,403
Total Pages	4,285	Registered Users	609
Upload Files	2,140	View Statistics	2,398,268

CSDMS COMMUNITY

There are 8 Working and Focus Research Groups, consisting of members from 130 US Institutions, 19 US Federal labs & agencies, 110 Foreign Institutes in 35 countries. The 556 CSDMS Members are distributed in the following **Working and Focus Research Groups** as of 07/31/11:

Terrestrial	**269**	**Cyber**	**104**
Coastal	**201**	**EKT**	**75**
Hydrology	**177**	**Carbonate**	**51**
Marine	**151**	**Chesapeake**	**38**

Participating U.S. agencies include: NSF, Office of Naval Research, Army Corps of Engineers. Army Research Office, U.S. Geological Survey, NASA, National Oceanic and Atmospheric Administration, National Oceanographic Partnership Program, Idaho National Laboratory, National Park Service, National Forest Service, U.S. Dept of Agriculture, EPA, Argonne National Laboratory, National Weather Service, Naval Research Laboratory, National Center for Atmospheric Research, Nuclear Regulatory Commission. A CSDMS Interagency Committee serves the function of both communication and coordination.

Industry Partners include: BHP Billiton Petroleum, Chevron Energy Technology, ConocoPhillips, Deltares, ExxonMobil Research and Engineering, Japan Agency for Marine-Earth Science & Technology (JAMSTEC), Schlumberger Information Solutions, Shell International, Petrobras, Statoil, and URS Corporation. These organizations collaborate via the participation of representatives in CSDMS committees and working groups, including a CSDMS Industrial Consortium.

CSDMS INTEGRATION FACILITY (IF)

The CSDMS Integration Facility (IF) maintains the CSDMS Repositories, facilitates community communication and coordination, public relations, and product penetration. IF develops the CSDMS cyber-infrastructure (e.g. coupling framework, tools, services and software protocols), and provides software guidance to the CSDMS community. CSDMS' IF is located at INSTAAR, University of Colorado-Boulder, csdms.colorado.edu/wiki/Contact_us. As of July 31, 2011, CSDMS IF staff included csdms.colorado.edu/wiki/Staff

- Executive Director, Prof. James Syvitski (April, 2007) — CSDMS & CU support
- Executive Assistant, Ms. Marlene Lofton (Aug. 2008) — CSDMS support
- Chief Software Engineer, Dr. Scott Peckham (April, 2007) — CSDMS & other NSF/NOAA support
- Software Engineer, Dr. Eric Hutton (April, 2007) — CSDMS & LASP & GSC support
- Software Engineer, Dr. Beichuan Yan (April, 2009- Aug 11) — CSDMS support --- *term ended*
- Computer Scientist, Jisamma Kallumadikal (Aug, 2009) — Industry, CSDMS & NOAA support
- Cyber Scientist Dr. Albert Kettner (July, 2007) — CSDMS, ConocoPhilips & other NSF support
- EKT Scientist Dr. Irina Overeem (Sept, 2007) — CSDMS, ConocoPhillips & other NSF support
- PDF Dr. Sagy Cohen (Aug, 2010) — NASA support
- Ph.D. GRA Stephanie Higgins (Sept, 2010) — Other NSF support
- Ph.D. GRA Fei Xing (July, 2010) — CSDMS & other NSF support
- Ph.D. GRA Ben Hudson (May, 2010) — NSF support
- Accounting Technician Mary Fentress (April, 2007) — multiple grant support
- Systems Administrator Chad Stoffel (April, 2007) — multiple grant support
- Director G Robert Brakenridge, Dartmouth Flood Observatory (Jan, 2010) — NASA support
- Senior Research Scientist Christopher Jenkins (Jan 2009) — NSF & other support

CSDMS VISITING SCIENTISTS AND STUDENTS since Jan 1, 2011:

- Zuosheng Yang Professor Ocean U of China 2011 January
- Houjie Wang Professor Ocean U of China 2011 January
- Naishuang Bi Professor Ocean U of China 2011 January
- Reed Maxwell Professor Col. School of Mines 2011 February
- Tao Sun Executive ExxonMobil 2011 March
- Damian O'Grady Executive ExxonMobil 2011 March
- Kim Picard Ph.D. student GSC, Pacific 2011 March-April
- Phillip Hill Fed Officer Geol. Survey of Canada 2011 March
- Cristen Torrey PDF CoG 2011 April
- Mohamad Nasr-Azadani Ph.D. student UCalifornia SB 2011 May
- *(CSDMS Student Modeler for 2010)*
- Laurel Saito Professor Univ Nevada-Reno 2011 June-2012
- Bert Jagers Executive Deltares 2011 June
- Kees Sloff Executive Deltares 2011 June
- Ron Tingook Ph.D. student U Alaska 2011 June
- Michael Barton Director Arizona State U 2011 June
- Liz Olhsson Ph.D. student UC Berkeley 2011 July
- Martin Perlmutter Executive Chevron 2011 July
- Michael Pyrcz Executive Chevron 2011 July
- Brian Willis Executive Chevron 2011 July

CSDMS IF PUBLICATIONS since Jan 1, 2011:

Book Chapters, Journal papers and Newsletters:

Submitted:

Campbell, K., Overeem, I., and Berlin, M. Taking it to the Streets: the Case for Modeling in the Geosciences Undergraduate Curriculum. *Computers & Geosciences.*

Cohen, S., Kettner, A.J., Syvitski, J.P.M., and Fekete, B.M., *submitted.* WBMsed: a distributed global-scale daily riverine sediment flux model -model description and validation. *Computers & Geosciences.*

De Winter, I., Storms, J., and Overeem, I. Glacial valley sediment budgets during deglaciation: A numerical sediment source module. *Geomorphology.*

Hutton, E.W.H., Syvitski, J.P.M., and Watts, A.B. Isostatic Flexure of a Finite Slope Due to Sea-Level Rise and Fall. *Computers & Geosciences.*

Kettner, AJ and Syvitski, JPM (Eds). Modeling for Environmental Change *A CSDMS Special Issue of 'Computers and Geosciences'.*

Overeem, I., Anderson, R.S., Wobus, C., Clow, G. D., Urban, F., and Matell, N. Quantifying the Role of Sea Ice Loss on Arctic Coastal Erosion. *Geophysical Research Letters.*

Peckham, S.D. and Goodall, J.L. Driving plug-and-play models with data from web services: A demonstration of interoperability between CSDMS and CUAHSI-HIS, *Computers and Geoscience.*

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

Peckham, S.D., Hutton, E.W.H., and Norris, B. A Component-Based Approach to Integrated Modeling in the Geosciences: The Design of CSDMS. *Computers & Geosciences.*

Upton, P., Kettner, A.J., Gomez, B., Orpin, A.R., Litchfield, N., and Page, M.J. Application of CSDMS codes to Source-to-Sink studies in New Zealand: The Waipaoa and the Waitaki catchments. *Computers & Geosciences.*

Accepted:

Chen, Y., Overeem, I., Syvitski, J.P.M., Gao, S., and Kettner, A.J. Controls of levee breaches on the Lower Yellow River during the years 1550-1855. *IAHS* Publ.

Foufoula-Georgiou, E., Syvitski, J., Paola, C., Chu Thai Hoanh, Phuc Tuong, Vörösmarty, C., Kremer, H., Brondizio, E., and Saito, Y. International Year of Deltas 2013 (IYD-2013): A Proposal, *Eos Forum*, accepted.

Maselli, V., Hutton, E.W., Kettner, A.J., Syvitski, J.P.M., and Trincardi, F. Evidence of high-frequency sea level and sediment supply fluctuations during Termination I: an integrated sequence-stratigraphy and modeling approach from the Adriatic Sea. *Marine Geology*.

Matell, N., Anderson, R. S., Overeem, I., Wobus, C., Urban, F. and Clow, G. Subsurface thermal structure surrounding thaw lakes of different depths in a warming climate. *Computers & Geosciences*.

McCarney-Castle, K., Voulgaris, G., Kettner, A.J., and Giosan, L. Simulating fluvial fluxes in the Danube watershed: The Little Ice Age versus modern day. *The Holocene*.

Overeem, I., Kettner, A.J., and Syvitski, J.P.M. Management and human effects., In: Wohl, E., (ed.), 2011.*Treatise of Geomorphology: Fluvial Geomorphology*.

Restrepo, J.D., and Kettner, A.J. Human induced discharge diversion in a tropical delta and its environmental implications: the Patía River, Colombia. *Journal of Hydrology*.

Slingerland, R., and Syvitski, J.P.M. Community Approach to Modeling Earth- and Seascapes. *Treatise on Geomorphology*, in press

Syvitski, J.P.M., Peckham, S.P., David, O., Goodall, J.L., Delucca, C., Theurich, G. Cyberinfrastructure and Community Environmental Modeling. In: *Handbook in Environmental Fluid Dynamics*, Editor: H.J.S. Fernando, Taylor and Francis Publ

Wobus, C., R.S. Anderson, I. Overeem, N. Matell, F. Urban, G. Clow, and C. Holmes. Calibrating thermal erosion models along an Arctic coastline. *Arctic Antarctic and Alpine Research*.

Published:

Christoffersen, P., R. Mugford, K.J. Heywood, I. Joughin, J.A. Dowdeswell, J.P.M. Syvitski, A. Luckman, and T.J. Benham, 2011. Warming of waters in an East Greenland fjord prior to glacier retreat: mechanisms and connection to large-scale atmospheric conditions, The Cryorsphere Discussions 5, 1335–1364, 2011 doi:10.5194/tcd-5-1335-2011

Kao, S.J., M. Dai, K. Selvaraj, W. Zhai, P. Cai, S.N. Chen, J.Y. Yang, J.T. Liu, C.C. Liu, and J.P.M. Syvitski, 2010. Cyclone-driven deep sea injection of freshwater and heat by hyperpycnal flow in the subtropics, Geophysical Research Letters 37, L21702, doi:10.1029/2010GL044893.

Pyles, D.R., Syvitski, J.P.M., and Slatt, R.M., 2011. Applying the concept of stratigraphic grade to reservoir architecture along the shelf-edge to basin-floor profile: an outcrop perspective, Marine and Petroleum Geology 28: 675-697. doi:10.1016/j.marpetgeo.2010.07.006

Syvitski, J.P.M., and Kettner, A.J., 2011. Sediment Flux and the Anthropocene. Philosophical transactions of the Royal Society, 369, 957-975, doi: 10.1098/rsta.2010.0329.

Syvitski, J.P.M., Hutton, EWH, Peckham, SD, and Slingerland, RL, 2011. CSDMS — A Modeling System to Aid Sedimentary Research. The Sedimentary Record 9, 1-9.

Syvitski, J.P.M., 2011. Global sediment fluxes to the Earth's coastal ocean. Applied Geochemistry 26 (2011) S373–S374

Ward D.J., M.M. Berlin, and R.S. Anderson (2011), Sediment dynamics below retreating cliffs. *Earth Surface Processes and Landforms*. DOI: 10.1002/esp.2129.

Abstracts since Jan 1, 2011:

Ashton, A., Giosan, L., Kettner, A.J., Hutton, E.H.W., and Ibanez, C., April 2011. Influence of wave angle distribution and sediment supply variation on plan-view delta morphology: application to the Ebro Delta, Spain. EGU, Vienna, Austria.

Hannon, M.T., Kettner, A.J., Syvitski, J.P.M., and Overeem, I., March 2011. Longitudinal profiles, Neotectonics, and Potential Bedload Transport. Hydrological Science symposium, Boulder CO., USA.

Hudson, B., Overeem, I., McGrath, D., Rick, U., Syvitski, J., and Zettlermann, A., March 2011. Sediment Plumes as proxy for melt on the Greenland Ice Sheet: Possible evidence for a long and intense 2010 melt season. Annual Arctic Workshop, Montreal, Canada.

Kettner, A.J., and Brakenridge, G.R., April 2011. Estimating time series of fluvial suspended sediment by applying remote sensing techniques. EGU, Vienna, Austria.

Kettner, A.J., Xing, F., Ashton, A., Hannon, M., Ibanez, C., and Giosan, L., April 2011. Unraveling the impact of humans versus climate on the morphological evolution of the Ebro Delta, Spain. EGU, Vienna, Austria.

Overeem, I., Syvitski, J., Kettner, A.J., Hutton, E., and Brakenridge, B., March 2011. Sinking Deltas due to Human Activities, Invited talk for Tulsa Geological Society. In: AAPG Search and Discovery #70094.

Overeem, I.; Hudson, B.; Berlin, M.; Mcgrath, D.; Syvitski, J.P.M.; and Mernild, S. Jan 24-27 2011. Fjord sediment plumes as indicators of west greenland ice sheet freshwater flux, Abstracts of the *AGU Chapman Conference on Source to Sink Systems around the world and through time.* Oxnard, CA, p. 55-56.

Peckham, S.D., July 2011. Component-based ocean modeling with the Community Surface Dynamics Modeling System (CSDMS), Chesapeake Bay Program (CBP) Modeling Quarterly Review Meeting, Annapolis, MD.

Peckham, S.D., June 2011. Component-based ocean modeling with the Community Surface Dynamics Modeling System (CSDMS), Chesapeake Community Modeling Program (CCMP) Hydrodynamic Modeling Workshop, Smithsonian Environmental Research Center (SERC), Edgewater, MD.

Rick, U., Abdalati, W., Overeem, I., Berlin, M., and van den Broeke, M., February 2011. Evidence for Substantial Englacial Retention of Surface Meltwater. IAG-workshop Mass balance of glaciers and icecaps, Presentation and abstract.

Syvitski, JPM, May 11th, 2011. The Anthropocene from land to sea. Abstracts of The Anthropocene: a new geological epoch? Geological Society of London, p. 7.

Syvitski, JPM, 02-04 March 2011. Deltas under climate change- the challenges of adaptation. Delta 2011: Deltas under climate change: the challenges of adaptation. Ha Noi, Vietnam.

Syvitski, J.P.M., 12-15 September 2011, Deltas under climate change- the challenges of adaptation. LOICZ Open Science Conference 2011: "Coastal Systems, Global Change and Sustainability". Yantai, China.

Syvitski, J.P.M., Jan 24-27, 2011, Source to Sink Numerical Modeling of Whole Dispersal Systems, Abstracts of the *AGU Chapman Conference on Source to Sink Systems around the world and through time, 2011*, Oxnard, CA, p. 71

Syvitski, J.P.M., June 6-10, 2011, The Anthropocene battleground: Geology, geography and human influence on the delivery of sediment to the coastal ocean. Abstracts of the Deltanet International Conference: Impacts of Global Change on Deltas, Estuaries and Coastal Lagoons, Research, observation and management. Ebro Delta, Catalonia, Spain, pg 46-47.

Syvitski, J.P.M., R.G. Brakenridge, and M.D. Hannon, Sept. 6~8, 2011. The Great Indus Flood of 2010, RCEM 2011: The 7th IAHR Symposium on River, Coastal and Estuarine Morphodynamics, Tsinghua University, Beijing, China

Upton, P., Litchfield, N., Orpin, A., Kettner, A., Hicks, M., and Vandergoes, M., January 2011. Modelling Source-to-Sink systems in New Zealand: The Waipaoa and Waitaki catchments. AGU Chapman Conference on Source to Sink Systems Around the World and Through Time, Oxnard, CA, USA.

Xing, F., Kettner, A.J., and Hannon, M.T., March 2011. Impact of Climate change and Human interference on the evolution of the Ebro Delta, Spain in the last 2000 years. Hydrological Science symposium, Boulder CO., USA.

Progress on Year 5 Goals (April – July, 2011)

Goal 1) CSDMS Web Gateway and Portal in Aid of Community Involvement

The CSDMS website is evolving at a rapid pace, maturing to become the portal for open source surface dynamics models, almost always ranking number one for Google searches on specific model names. Several new content management developments have taken place over the last half-year to become and stay *the* portal for open source surface dynamics models. Listed below are this year major achievements to serve our community.

A system to decide on the transitions of models into fully integrated components. *(Completed)*
Receiving feedback from the CSDMS community regarding which model should be incorporated in the CMT is of utmost importance. In the past, this information was obtained during various WG and FRG meetings. However the disadvantages were that 1) only attendants had a say, 2) decisions where not anonymous, 3) not everybody has felt confident to share his or her thoughts during a public meeting, and 4) the WG and FRG meetings are only held once a year so a desire to incorporate a model into the CMT (CSDMS's Component Modeling Tool) could only be expressed once a year.

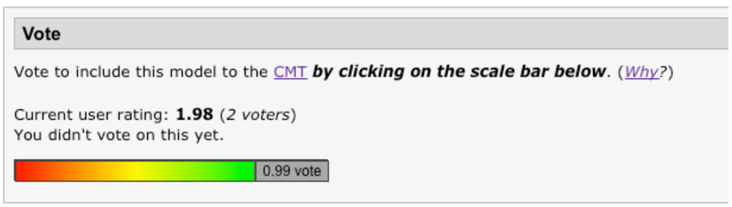


Figure 1. An example of the voting tool displayed on each model description page.

To overcome these disadvantages, a smart online voting system as decision tool was implemented to prioritize modules. Smart in a sense that the voting system only allows *CSDMS members* to express their needs on which model to incorporate, and they only receive *one* vote per model. To express a need through voting, a member has to log in to the website and go to a specific model description. At the top of each model description page a voting menu is displayed when the model is not yet incorporated within CMT (Fig. 1). BY simply clicking on the scale bar to express your vote, the vote is registered. Voting results are publicly displayed on each model description page as well as listed in real time in a model overview page (Fig.1 & 2). The voter stays anonymous. A vote can range from 0 to 1, where 0 means no need to incorporate this model in the CMT and 1 means a high desire to incorporate the model. The vote can be changed at any time up to the point where action is taken by CSDMS-IF to start integrating the specific model. A brief guideline on the voting process is provided as well.

Program	Description	Developer	Voting results	Download
SedBerg	An iceberg drift and melt model, developed to simulate sedimentation in high-latitude glaciated fjords.	Mugford, Ruth	1.98 (2 voters)	
XBeach	Wave propagation sediment transport model	Roelvink, Dano	1.86 (2 voters)	
MODFLOW	MODFLOW is a three-dimensional finite-difference ground-water model	Barlow, Paul	1.74 (2 voters)	
Caesar	Cellular landscape evolution model	Coulthard, Tom	1.5 (2 voters)	
Delft3D	3D hydrodynamic and sediment transport model	Delft3D, Support	1.5 (2 voters)	
Anuga	ANUGA is a hydrodynamic modelling tool that allows users to model realistic flow problems in complex 2D geometries.	Habili, Nariman	1 (1 voter)	
GOLEM	Landscape evolution model	Tucker, Greg	1 (1 voter)	
PIHM	PIHM is a multiprocess, multi-scale hydrologic model.	Duffy, Christopher	1 (1 voter)	
RHESSys	Regional Hydro-Ecologic Simulation System	Tague, christina	1 (1 voter)	
SIBERIA	SIBERIA simulates the evolution of landscapes under the action of runoff and erosion over long times scales.	Willgoose, Garry	1 (1 voter)	
SWAN	SWAN is a third-generation wave model	SWAN, Team	1 (1 voter)	
WILSIM	Landscape evolution model	Luo, Wei	1 (1 voter)	
OTEQ	One-Dimensional Transport with Equilibrium Chemistry (OTEQ): A Reactive Transport Model for Streams and Rivers	Runkel, Rob	0.99 (1 voter)	
ParFlow	Parallel, high-performance, integrated watershed model	Maxwell, Reed	0.99 (1 voter)	

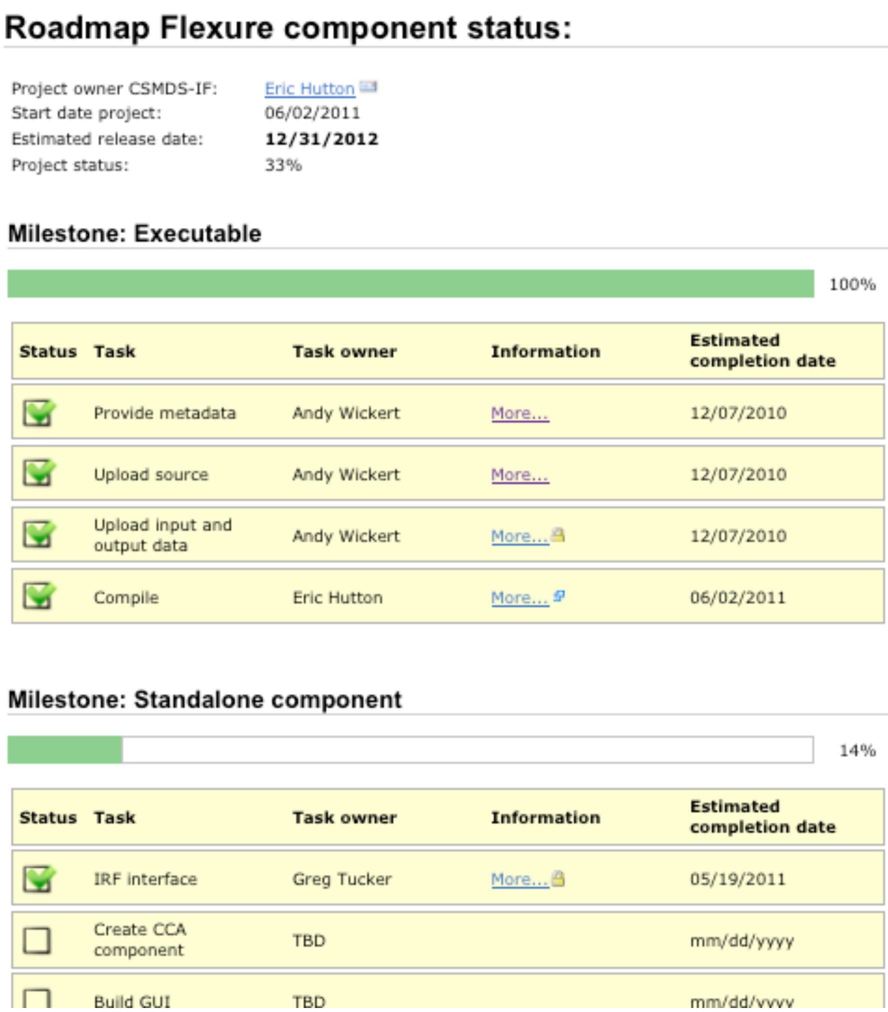
Figure 2. An example display of the voting results (column 4) in the model description list http://csdms.colorado.edu/wiki/Models_all

Members are encouraged to give their feedback on which model to incorporate into the CMT by: 1) advertising the online voting tool on the front-page of the CMDMS website, 2) informing the WG chairs of this new online feature, and 3) through email lists which were sent by the WG chairs.

Links:

- Voting guidelines: http://csdms.colorado.edu/wiki/Why_vote_for_model_incorporation
- Voting results: http://csdms.colorado.edu/wiki/Models_all
- Example voting box: http://csdms.colorado.edu/wiki/Model:ADCIRC

CSDMS development tracking: Roadmap to component status *(Completed)*
A roadmap displaying duration, tasks and person responsible, is automatically generated, to be filled out by a CSDMS-IF project owner once it is decided to be incorporate a model into the CMT. The roadmap is constructed such that it is easy to get a quick overview of the status of the project and contains the option for each of the task owners as well as for the project owner to incorporate links containing detailed information regarding specific tasks. An example link would be to a file that contains detailed information on how to compile the model source code on the CSDMS HPC (Fig. 3).



Roadmap Flexure component status:

Project owner CSMDS-IF:	Eric Hutton
Start date project:	06/02/2011
Estimated release date:	**12/31/2012**
Project status:	33%

Milestone: Executable

100%

Status	Task	Task owner	Information	Estimated completion date
✓	Provide metadata	Andy Wickert	More...	12/07/2010
✓	Upload source	Andy Wickert	More...	12/07/2010
✓	Upload input and output data	Andy Wickert	More...	12/07/2010
✓	Compile	Eric Hutton	More...	06/02/2011

Milestone: Standalone component

14%

Status	Task	Task owner	Information	Estimated completion date
✓	IRF interface	Greg Tucker	More...	05/19/2011
☐	Create CCA component	TBD		mm/dd/yyyy
☐	Build GUI	TBD		mm/dd/yyyy

Figure 3. The roadmap for the Flexure model, describing the project status of componentizing *http://csdms.colorado.edu/wiki/Roadmap:Flexure*

Three milestones, including their status are also displayed: executable, standalone component and coupled component. A green checkmark is placed when a task is fulfilled; a red cross is displayed when a task could not be executed. A task is displayed as light gray in cases where this task will not be fulfilled within the scope of the project; not every model will be configured as a component that can be coupled. With the roadmap in place we hope to inform our members about the status of a model to become a CMT component and provide detailed information of each of the involved tasks and which person to contact in case members have specific questions.

Links:
- Roadmap example: http://csdms.colorado.edu/wiki/Roadmap:Flexure

CSDMS' YouTube channel for educational movies, tutorial and model animations. *(Ongoing)*
CSDMS has ported all of its contributed animations and movies to a more publically used media, YouTube. This was executed to enlarge the impact of the community and expose the public to some of the community gained insights. Detailed description of each of the movies remain on the CSDMS website, under the educational section. While movies will still play from the CSDMS website they are hosted from the 'CSDMSmovies YouTube channel' (http://www.youtube.com/user/CSDMSmovie). The channel incorporates 7 playlists: Coastal animations (*21*), Environmental animations (*8*), Laboratory movies (*13*), Marine animations (*9*), Real event movies (*31*), Terrestrial animations (*16*), and CSDMS tutorials (*4*). In 2011, the University of Colorado started to encourage departments and institutes to provide animations and movies to the university media page as well. CSDMS contributed all its movies to CU to further enlarge the exposure to the public.

Below are some YouTube statistics after being operational for 7 months (channel went live on December 29th, 2010):
Nr. of movies & animations on the CSDMS YouTube channel: 98
Total views: 13,692 (~140 views per movie or animation)

Table 1: Top 10 views of CSDMSmovies YouTube channel:

Movie / animation description	**Nr. of views over a 7 month period**
Global circulation	2,137
Delta formation	992
Spit evolution	758
Jokulhlaup over Sandur Iceland	517
Sand boil behind levee	488
Sand ripples	429
Arctic coastal erosion 2010	389
Levee breach	361
Glacier surge	320
Lauren tide Ice Sheet evolution	287

The goal to enlarge the impact of the community by making the movies more accessible seems successful. **The CSDMS movies YouTube channel has been highlighted several times for being in the "Top 50 most viewed channel" of the "non profit" category.**

Links:
- Movie descriptions: http://csdms.colorado.edu/wiki/Movies_portal
- CSDMS YouTube channel: http://www.youtube.com/user/CSDMSmovie
- Univ. of Colorado YouTube channel: http://www.youtube.com/user/univcoloradoboulder#p/c/0A49CA0F0E6D8EDA

Tools for repository downloads embedded into the website is now open-access. *(Completed)*
Significant changes have been made on the backside of the model repository to accommodate community members desire to: 1) store and retrieve all source code of modules that are in the CSDMS database from a single place, 2) track basic information of who is downloading what module from the CSDMS database and

3) monitor how often a module is downloaded from the CSDMS database.

To achieve this all source code is now only stored in Subversion. People who download a module access subversion automatically through the website, select the desired version of the source code of a module, which then is again automatically zipped before the download process starts. We do solicit each downloaders email address and name (Fig. 4). This collected information will be provided to the original developer on request.

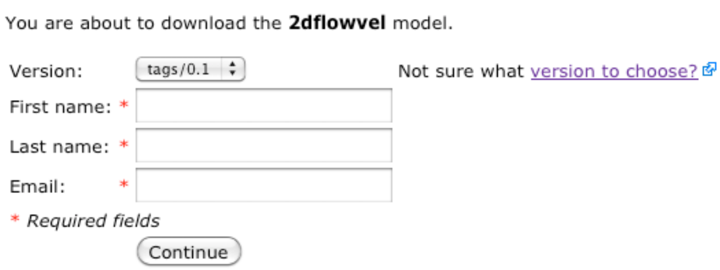


Figure 4. Model download menu example where you can select the desired version of a model as well as provide basic information. *http://csdms.colorado.edu/wiki/Special:ExtensionDistributor*

Monthly download statistics are presented on the model metadata webpage as soon as a module is downloaded once (Fig. 5). Complete download statistics of the model repository are provided as well (see links below).

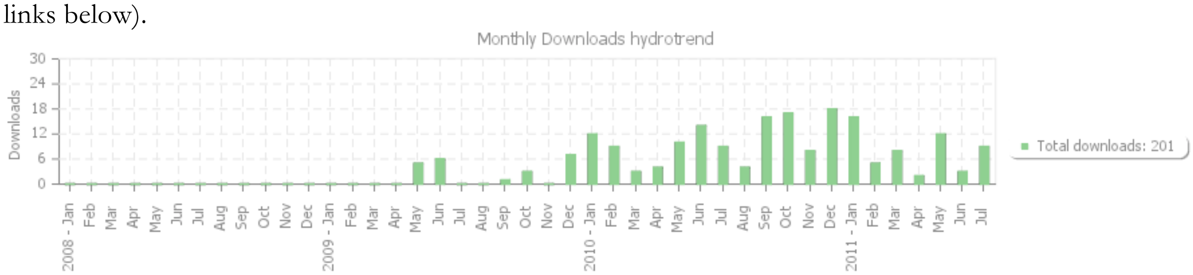


Figure 5. Example of monthly download statistics, made available for each model in the CSDMS repository.

<u>Links</u>:

- Download a model: http://csdms.colorado.edu/wiki/Download_models
- Monthly overview of a model download e.g.: http://csdms.colorado.edu/wiki/Model:SIBERIA
- Complete download report: http://csdms.colorado.edu/wiki/Model_download_Page

CSDMS HPC (beach) use has become open-access *(Completed)*

CSDMS uses Ganglia, a scalable distributed monitoring system, to monitor beach, the high-performance cluster of CSDMS. Real-time monitoring information is of key value for cluster operators but can also be very relevant for its users. Therefore CSDMS decided to integrate key output parameters of ganglia into the CSDMS website. Visitors can monitor status and activity of the cluster as a whole as well as of each of the nodes (Fig. 6). A ganglia summary is posted real-time on the front CSDMS website under 'Supercomputing stats' as well.

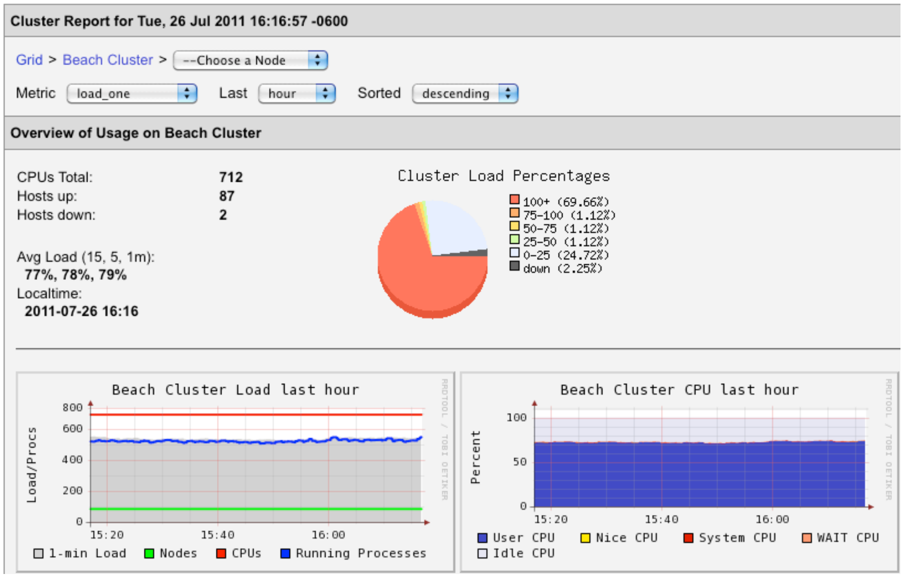
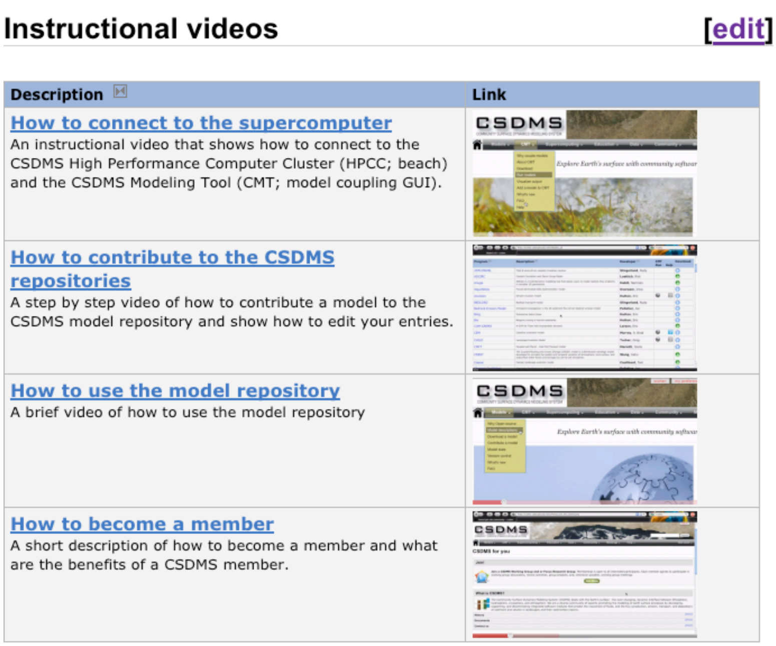


Figure 6. Snapshot in time of the use of Beach, provided by Ganglia. http://csdms.colorado.edu/wiki/HPCC_current_use

Links:
- Integrated ganglia page: http://csdms.colorado.edu/wiki/HPCC_current_use
- Summary of ganglia on front page: http://csdms.colorado.edu

Video tutorials on topics related to modeling with the CSDMS Modeling Tool (CMT). *(Completed)* CSDMS members are exposed to a lot of content that at a first glance seems difficult or time consuming to achieve comprehension. Topics are well explained in written documents and posted on the community website, but have been either difficult to find if the user doesn't know where to look for them, or the user simply does not have the time to read all instructions, which eventually results in reduced participation of the community. To increase participation, four video tutorials are developed to make CSDMS processes more comprehensible for our members: 1) How to connect to the CSDMS HPC, 2) How to contribute to the CSDMS repositories, 3) How to use the model repository, and 4) How to become a member (Fig. 7).



Instructional videos [edit]

Description	Link
How to connect to the supercomputer An instructional video that shows how to connect to the CSDMS High Performance Computer Cluster (HPCC; beach) and the CSDMS Modeling Tool (CMT; model coupling GUI).	
How to contribute to the CSDMS repositories A step by step video of how to contribute a model to the CSDMS model repository and show how to edit your entries.	
How to use the model repository A brief video of how to use the model repository	
How to become a member A short description of how to become a member and what are the benefits of a CSDMS member.	

Figure 7. List of the available tutorial videos. http://csdms.colorado.edu/wiki/Help:How_to_videos

The tutorial videos (posted on the CSMDS YouTube channel) are embedded in the CSDMS website and are between 2:30 and 8 minutes long, taking the user step by step through a particular process. The videos are featured under the "Help" menu on the main menu bar of the website as well as embedded on pages that describe a specific process.

Links:
- How to videos: http://csdms.colorado.edu/wiki/Help:How_to_videos
- CSDMS movie channel: http://www.youtube.com/user/CSDMSmovie

New model or science in the spotlight as highlighted on the CSDMS front page. *(Ongoing)*
CSDMS launched its new web portal last December. The new web portal aims to enthuse, inform and engage end-users by more frequent updates on CSDMS science and new discoveries. Two sections, 'Model highlight' and 'Science in the spotlight' are embedded at the front page of the CSDMS website for this purpose. Each section provides a summary of a topic with a link to the full article. So far 7 topics (See table 2) have been featured generating (up to July 26th) in total 2,235 hits.

Table 2: Recent Model highlights and Science in the Spotlight topics

Model highlight (*1,250 views*)	Science in the spotlights (*985 views*)
TopoFlow	Boom-and-bust cycles of barrier island retreat
TURBINS: An immersed boundary, Navier-Stokes code for the simulation of gravity and turbidity currents	Retreating Arctic Coasts
Delft3D	Where do Salmon thrive
SedBerg	

Links:
- Entrance page CSDMS: http://csdms.colorado.edu
- Model highlight history: http://csdms.colorado.edu/wiki/Model_highlight
- Science in the spotlight history: http://csdms.colorado.edu/wiki/Science_spotlights

CSDMS will actively share news through social networking; Twitter. *(Ongoing)*
A twitter account has been set up to reach out within and beyond our community (Fig. 8). Several options (wiki external plugins) has been investigated to incorporate the provided 'tweets' within the CSDMS website for users to view older tweets as well. Providing new tweets and a fully integration of old tweets into the CSDMS website will be one of the targets for the second half of this year.
Links:
- Twitter page of CSDMS: http://twitter.com/#!/CSDMS

Figure 8. CSDMS is 'tweeting'. *http://twitter.com/#!/CSDMS*

Google Analytics to monitor key web-use parameters is integrated into the CSDMS website. *(Ongoing)* Google Analytics content management monitoring software informs on how people touch upon and explore the CSDMS website. With this information we analyze which pages are most often viewed, how people reached those pages, which pages are more buried and hard to find by the user, and where we should place content that needs visibility. The monitoring software has been integrated within the CSDMS website since January 8th, 2010. Some of the results we would like to share with our users by integrating key parameters monitored by Google Analytics into the CSDMS website. Several options have been explored for this integration. Third party software (e.g.: http://www.embeddedanalytics.com/) is available to fulfill this need but limited in usability within the used content management software. The free available Google Analytics Management API as well as the Data Export API (http://code.google.com/apis/analytics/docs/) seems to be better adaptable for this purpose and the integration of these tools will be one of the targets for the second half of this year.

CSDMS Communication Strategy

CSDMS has a goal to continually increase its profile within relevant research, educational and industrial communities both nationally and internationally. CSDMS has a diverse membership and works to develop targeted communication with each audience. CSDMS is responsible to continually interact with its community so as to address real community needs (i.e., expansive CSDMS standalone model repository, componentization offerings). In doing so, CSDMS intends to continually refine its processes and facilitate leading edge science involving Earth surface dynamics modeling. Through all the methods below, we intend to continue to gather strategic information from our community and adapt our services to meet their needs to the best of our ability and within our budgetary and time constraints.

CSDMS Interactive Website Examples

- CSDMS website profiles our models, member scientists and their work (model highlights)
- CSDMS website posts jobs available within the community
- CSDMS website profiles upcoming meetings within the community
- CSDMS website highlights relevant science (science in the spotlight)
- Video tutorials on how to use CSDMS wiki website and interact with CSDMS model repository

CSDMS Meetings.

- **Working Group (WG) and Focus Research Group (FRG) Workshops** Each group has a

Chairperson who corresponds to his/her membership via telephone, meetings, and mail lists. The WG and FRG Chairperson and CSDMS IF staff conduct polls of the WG and FRG membership to prioritize the work of CSDMS, which helps to prioritize CSDMS operations budget allocation.

- **Annual Meeting**: **"CSDMS Meeting 2010: Modeling for Environmental Change"** In 2010, the first 'all-hands meeting' was held in San Antonio Texas. This meeting allowed CSDMS members to share their feedback with CSDMS during meetings, presentations, question and answer sessions, email, and feedback survey forms. The feedback was consolidated and suggestions were incorporated into the WG and FRG future goals and the format/content of the upcoming **"CSDMS Meeting 2011: Impact of Time and Process Scales"** to be held October 28-30, 2011 in Boulder, CO.

- **CSDMS Inter-Agency Meeting**. CSDMS provides updates to U.S. agencies.

Personal Interviews with Key Personnel

- **CoG Interviews**. The Commodity Governance (*Introducing commodity governance into community Earth science modeling)* or COG is a type II NSF/CDI project to research communication strategies and built software tools to enable virtual organizations in the Earth Sciences like CSDMS to scale to massive interdisciplinary "communities of communities." COG interviews were held with CSDMS staff and with volunteer scientists, government users and students within the larger CSDMS community. The results are being compiled and analyzed with a goal towards publication to provide further insight into how best to communicate and strategize for a diverse community that mainly interacts via the virtual world (i.e., wiki website, teleconferences, email lists, discussion forums).

Survey on newly launched web portal

- CSDMS IF staff requested feedback on scope, clarity, content, useability and navigation, and aesthetics of the newly launched CSDMS wiki from web professionals and science institute data managers, students, as well as from the EKT Working Group Chair and key members of the CSDMS steering committee in January-February 2011. Responses were overwhelmingly positive and suggested additional changes have mostly been implemented by July 2011.
  One reviewer stated:" **It looks good and a big improvement. You clearly spent some time in the redesign. Creating a decent website is not easy... congrats to all involved!"**

CSDMS Integration Facility Staff publications and presentations

- CSDMS IF staff promote CSDMS and stay current with the latest industry information by conducting research published in leading venues (publications list provided above) and providing key educational presentations and mini-courses (world-wide) and at CSDMS co-sponsored meetings (meetings list included above)

CSDMS Student Modeler-of-the-Year Contest

- The **CSDMS Student Modeler Award** is an annual competitive award for graduate students from Earth and computer sciences who have completed an outstanding research project involved in developing an Earth science model (terrestrial, coastal, marine or biogeochemistry), a modeling tool or model linking technology. Entries are judged by a panel of experts in the field on the basis of ingenuity, applicability, and contribution towards the advancement of geo-science modeling. This award increases the recognition of CSDMS within the graduate student population and their institutions. Winners receive a funded visit to the CSDMS Integration Facility in Boulder, Colorado, to learn and work with CSDMS scientists to develop their model into a CSDMS component.

Missives from the Executive Director

- Missives from the Executive Director of CSDMS are sent to every member highlighting progress, news, and membership events. Once quarterly, these missives have decreased to 2 - 3 times a year, in lieu of increased social and wiki communication. Email is an overused communication forum

Social Marketing

- CSDMS has a presence on Twitter

Goal 2) Componentizing the CSDMS Model Repository

Significant progress has been made in the last 6 months on componentizing CSDMS models. This section summarizes the specific progress that has been made to date on the following tasks.

Regional Ocean Modeling System-ROMS Builder. ROMS differs from most models in our repository in that each user creates and compiles their own, customized version of ROMS, based on the science questions involved and the module options one needs. Recognizing this, CSDMS has created a component we call "ROMS Builder" that allows a user to perform this task within the graphical user interface of the CSDMS Component Modeling Tool and then wraps the resulting executable as a component that can be used within the CSDMS CMT and that automatically appears in the pallette. ROMS Builder was tested by CSDMS member Aaron Bever (UMCES) and improved based on his feedback.

ChesROMS, UMCES_ROMS and CBOFS2. On specific request of the Chesapeak Focus Research group, ROMS Builder has been used to create componentized versions of three key instances of ROMS. Each has a different spatial resolution and is used for modeling the Chesapeake Bay.

LTRANS- The Larval TRANSport Lagrangian model (LTRANS) is an off-line particle-tracking model that runs with the stored predictions of a 3D hydrodynamic model, specifically the Regional Ocean Modeling System (ROMS). CSDMS has worked with the developers of LTRANS to create a version 2 that is much more efficient and that exposes the basic IRF interface. The new version appears in the CMT and CSDMS is in the process of testing it. Modifications to permit oil spill tracking have also been made and will be available in the next release.

MARSSIM. A landform evolution model operating at the drainage basin or larger scale. This landscape evolution model can now be run through the CMT, has a tabbed-dialog GUI and has passed a series of test cases.

Note: ROMS, LTRANS and MARSSIM are each written in Fortran and CSDMS is working on a unified approach to providing the getter function that is required for each them.

Flexure. This flexural and non-flexural isostasy model provides 1D and 2D solutions. Flexure is the first model submitted by a new graduate student, who fully committed to help bring the model code online as a component in the CMT. Flexure has been refactored to provide the IRF interface and is very close to appearing as a plug-and-play component in the CMT. It will have many coupling options in both the terrestrial model projects as well as in the coastal and marine model projects. This model has strong interest from CSDMS industry partners to allow coupling applications with stratigraphic models.

Bioenergetics. This is a biological model with a large user base that was originally developed by Paul Hanson of the University of Wisconsin Center for Limnology. It uses an energy balance formulation to compute the growth potential of different fish species as a function of environmental variables such as water temperature. CSDMS member Laurel Saito (and developer of open-source extensions to the model) has recently obtained permission to provide the full model and its documentation as a set of plug-and-play components. She is working with CSDMS to (1) determine how best to break the model into a set of reusable, plug-and-play components, (2) refactor the components with the IRF interface and (3) convert the model's documentation to HTML.

ParFlow. ParFlow is an open-source, object-oriented, parallel watershed flow model. It includes fully-integrated overland flow, the ability to simulate complex topography, geology and heterogeneity and coupled land-surface processes including the land-energy budget, biogeochemistry and snow. ParFlow uses a TCL framework to integrate its various components. CSDMS is studying the source code to determine whether its engine (a Richards equation solver) can be provided as a separate plug-and-play component. Some code changes by the developer may be necessary to achieve this.

Grid generator/editor. Several models in the CSDMS repository require a computational mesh for the area to be modeled but they rely on external software for this preliminary step. CSDMS has surveyed existing, open-source software for grid generation. GridGen and Triangle appear to meet our needs and we have identified an interactive, graphical front-end for GridGen written in Java (but not yet complete). CSDMS will determine if it is feasible to provide this within the CMT.

Erode3. Erode is a raster-based, fluvial landscape evolution model. This model provides all of the CSDMS interface functions and will be made available within the CMT soon. Erode has undergone a first pass by graduate student testers.

CUAHSI HydroModeler Suite. These process modules each have a simplified OpenMI interface which simplifies their inclusion in the CSDMS framework. However, some of them are written in C#, which though similar to Java is not a Babel-supported language. In addition, funding for the CUAHSI HydroDesktop project is uncertain, so it is not yet clear whether CMT can be provided within HydroDesktop. CSDMS will determine how best to proceed over the remainder of this year.

Carbonate Workbench. The Carbonate Focus Research Group has made significant progress and plug-and-play components are expected later this year.

We intend to work with model developers to componentize the following models in the second half of this year: AquaTellus (Irina Overeem) and mARM4D (Sagy Cohen).

Goal 3) Advancing Selected Goals of the Working Groups & Focus Research Groups

As described in the previous section, significant progress has been made in converting the specific models identified by the working groups as CSDMS plug-and-play components.

CSDMS has defined a Basic Model Interface or BMI that is to be provided by model developers and a Component Model Interface or CMI for model coupling that is provided by CSDMS. CSDMS continues to improve its automated tools that wrap BMI-compliant models with the CMI interface. CSDMS has produced draft documents that describe these two interfaces in detail and will soon finalize them. When finalized, language-specific versions will be adapted for each of the Babel-supported languages. In addition, a paper has been submitted to a special issue of Computers and Geosciences that describes the inner workings and rationale of the CSDMS design.

CSDMS now provides a THREDDS Data Server that provides members with convenient web access to various data sets including, for example, the netCDF history files (model output) for the ROMS ocean model. This resource is currently being used to archive and share data from the U.S. Integrated Ocean Observing System (IOOS, ioos.gov) Modeling Testbed project.

Goal 4) Conferences, Meetings, and the 2nd CSDMS Special Issue

STAFF PARTICIPATION — CONFERENCES & MEETINGS

*01/2011	AGU Chapman Conf. Source to Sink	Oxnard, CA	(Overeem, Syvitski)
01/2011	Community for Integrated Env. Modeling (CIEM)	teleconferences	(Peckham)
02/2011	EPSCoR Climate IWG	McCall, Idaho	(Peckham)
02/2011	IASC Network for Arctic Glaciology	Winter Park, CO	(Overeem)
02/2011	WHOI Geodynamics Lecture	Woods Hole, MA	(Syvitski)
02/2011	ONR Delta Meeting	Arlington, VA	(Syvitski, Brakenridge)
02/2011	IGBP SC Meeting	Washington, DC	(Syvitski)

02/2011	Community for Integrated Env. Modeling (CIEM)	teleconferences	(Peckham)
02/2011	IWMI Delta 2011: Deltas under climate change	Hanoi, Vietnam	(Syvitski)
03/2011	Tulsa Geological Society Presentation	Tulsa, OK	(Overeem)
03/2011	CUAHSI CHyMP Meeting	Irvine, CA	(Peckham)
03/2011	41st Arctic Workshop at Universite de Quebec	Montreal, Canada	(Hudson)
03/2011	CU Hydrological Symposium	Boulder, CO	(Hannon, Xing)
03/2011	Hydrologic Model Intercomparison Workshop	Golden, CO	(Peckham)
03/2011	BOEMRE teleconference	(Arango, Harris, Meiburg, Syvitski)	
04/2011	European Geosciences Union (EGU)	Vienna, Austria	(Kettner)
04/2011	Deltares OS Collaboration meeting	Delft, Netherlands	(Kettner, Overeem)
04/2011	KORDI, KOPRI, KNU: CSDMS Modeling Course	Korea	(Syvitski)
04/2011	Community for Integrated Env. Modeling (CIEM)	teleconferences	(Peckham)
05/2011	Chesapeake FRG Mtg at SERC	Baltimore, MD	(Peckham)
05/2011	Lamont-Doherty Colloquium	Palisades, New York	(Syvitski)
05/2011	British Geol. Society: The Anthropocene	London, UK	(Syvitski)
05/2011	11th International Coastal Symposium	Szczecin, Poland	(Syvitski)
05/2011	CSDMS Executive Committee Meeting	Boulder, CO	(IF Staff)
05/2011	BOEMRE Teleconference	(Arango, Harris, Meiburg, Syvitski)	
06/2011	Geochemistry of the Earth Surface	Boulder, CO	(Syvitski)
06/2011	DeltaNet: Impacts of Global change	Ainsa, Spain	(Syvitski)
06/2011	Commodity Governance Meeting at NOAA	Boulder, CO	(Syvitski, Overeem)
*06/2011	CCMP Hydrodynamic Model Wkshp (SERC)	Edgewater, MD	(Peckham)
06/2011	BOEMRE teleconference	(Arango, Harris, Meiburg, Syvitski)	
07/2011	CBP Modeling Quarterly Review Mtg	Annapolis, MD	(Peckham)
07/2011	BOEMRE Teleconference	(Arango, Harris, Meiburg, Syvitski)	
08/2011	NCED Summer Course	Minneapolis, MN	(Overeem)

** CSDMS co-sponsored meeting*

CSDMS Meeting 2011: Impact of Time and Process Scales (ongoing)

Plans continue for the all hands CSDMS 2011 Meeting 'Impact of Time and Process Scales' in Boulder, CO (Oct. 28-30). **Theme:** The **Impact of time and process scales** is this year's theme with emphasis on standalone surface dynamics models. Our theme on time and space addresses the software subtleties at the heart of all surface dynamic modeling efforts — whether landscape-evolution, morphodynamics or transport of material. How each of us deals with issues of time and space should be educational. Through keynote presentations, posters, and hands-on clinics, our community contributed standalone models will take the limelight. Of course advances in the Component Model Tool (CMT) and other supporting tools will also be represented. Break out sessions will allow our Working and Focus Research Groups to examine their activities with a future view.

CSDMS Special Issue of 'Computers and Geosciences' (ongoing)

Submitted and positively reviewed manuscripts by at least 2 reviewers:

1. *Ashton, A.D., Hutton, E.W.H., Kettner, A.J., Xing, F., Giosan, L.* Progress in Coupling Coastline and Fluvial Dynamics. ******
2. *Burgess, P.* CarboCAT: A Cellular Automata Model of Heterogeneous Carbonate Strata
3. *Campbell, K., Berlin, M., and Overeem, I.* Taking it to the Streets: the Case for Modeling in the Geosciences Undergraduate Curriculum.
4. *Cohen, S., Kettner, A.J., Syvitski, J.P.M., and Fekete, B.M.* WBMsed: a distributed global-scale riverine sediment flux model - model description and validation.
5. *Dunlap, R., Rugaber, S., and Mark, L.* A Feature Model of Coupling Technologies for Earth System Models.

6. *Hutton, E.W.H., Syvitski, J.P.M., and Watts, A.* Isostatic Flexure of a Finite Slope Due to Sea-Level Rise and Fall. *
7. *Lorenzo-Trueba, J., Voller, V.R., and Paola, C.* A geometric model of sediment delta dynamics under base-level change.
8. *Matell, N., Anderson, R.S., Overeem, I., Wobus, C., Urban, F.E., and Clow, G.D.* Modeling the subsurface thermal impact of Arctic thaw lakes in a warming climate.
9. *Murray, B., Gopalakrishnan, S., Smith, M.D., and McNamara, D.E.* Coupling Models of Human and Coastal Landscape Change.
10. *Nasr-Azadani, M. M., Hall, B., and Meiburg, E.* Polydisperse turbidity currents propagating over complex topography: Comparison of experimental and depth-resolved simulation results.
11. *Peckham, S.D., and Goodall, J.L.* Driving plug-and-play models with data from web services: A demonstration of interoperability between CSDMS and CUAHSI-HIS.
12. *Peckham, S.D., Hutton, E., and Norris, B.* A Component-Based Approach to Integrated Modeling in the Geosciences: The Design of CSDMS. **
13. *Upton, P., Kettner, A.J., Gomez, B., Orpin, A.R., Litchfield, N., and Page, M.J.* Simulating post-LGM riverine fluxes to the coastal zone: The Waipaoa catchment, New Zealand. *
14. *Villaret, C., Hervouet, J.-M., Kopmann, R., and Merkel, U.* Morphodynamic modelling using the Telemac finite-element system.
15. *Viparelli, E., Lauer, W., Belmont, P., and Parker, G.* A Numerical Model to Develop Long-term Sediment Budgets Using Isotopic Sediment Fingerprints.
16. *Yeh, T.-H., and Parker, G.* Matlab-based Software for Evaluating Sediment-Induced Stratification in Open-Channel Flows.

* *Not all reviews have been received by the main author yet.*
** *Submission is pending.*

Goal 5) Technical Advances in the CSDMS Cyber-Infrastructure

CSDMS staff is working on a suite of cyber issues to aid the future direction of the CSDMS modeling environment. Focus is on streamlining the component wrapping process for model developers, and opening up component generation to end-users of *CMT*.

Milestone 1: The CSDMS integration facility has developed a suite of tools that extends the CCA bocca utility. Included in this collection is *bocca-clone*, a command-line utility that wraps a model as a CSDMS-CCA component for use within the CSDMS-CCA modeling framework. The model must expose the appropriate IRF interface (along with value getters and/or setters), with details of the model's interface and how it has been installed on the target platform described in a configuration file (eg. lists of exchange items, names of interface functions, paths to shared libraries, etc.). The bocca-clone tool has been tested for use with C and C++ components but has yet to be used with the remaining CCA-supported languages.

Links:

- Subversion repository: http://csdms.colorado.edu/viewvc/bocca_tools/trunk/scripts/

Milestone 2: Through the CSDMS Component Modeling Tool (CMT), users are now able to run components that themselves create new components. As proof-of-concept, these so-called component factories have been used to create new components based on a Regional Ocean Modelling System (ROMS) component. To create the new component, the component factory downloads, compiles, and installs a new version of the model on the CSDMS cluster, *beach*. The model is built to the specifications of the user as provided by configuration menus in the CMT. The component factory then goes on to auto-generate the wrapping code necessary to create a usable component within the CSDMS modeling framwork. Following this process, the user now is able to use this new component within the CMT.

Links:

- Subversion repository: http://csdms.colorado.edu/viewvc/component_builder/trunk

Milestone 3: In support of Milestone 2, and described above, the CSDMS IF has created a component that is able to download, compile, and install software on the CSDMS HPC cluster for use outside of the CSDMS modeling framework.

Milestone 4: The CSDMS IF has created several new service classes that equip components with tools to manage common tasks such as the printing of output data, and IRF-port management.

IRFPortQueue. The IRFPortQueue class manages the IRF uses-ports of a component. This service class manages the connection and disconnection of a component's IRF ports, controls the exectution of each port's initialize, run and finalize functions, as well as grid mapping of the get_value functions.

PrintQueue. The PrintQueue class manages the printing of uniform rectilinear and non-uniform gridded data. The class writes uniform rectilinear grids to NetCDF files (the NCRasterFile class), and non-uniform meshes to VTK files (the VTKFile class). The service class also manages printing intervals for components when these intervals may not be the same as a component's time step.

ESMFRegrid. The ESMF field regridding operation moves data between fields that lie on different grids for the purpose of model coupling through a sparse matrix multiply interpolation between source field and destination grid. The ESMF regridding module has been componentized and will work as a service component within CMT. An algorithm for automating parallel partitioning unstructured mesh of randomly distributed triangulars has been implemented and tested to improve regridding performance.

Links:

- CSDMS components:
  - http://csdms.colorado.edu/viewvc/components/trunk/import/csdms/components/edu.csdms.tools.IRFPortQueue/
  - http://csdms.colorado.edu/viewvc/components/trunk/import/csdms/components/edu.csdms.tools.PrintQueue/
- Python modules:
  - http://csdms.colorado.edu/viewvc/cmt_py_utils/trunk/cmt/port_queue.py
  - http://csdms.colorado.edu/viewvc/cmt_py_utils/trunk/cmt/print_queue.py

Milestone 5: Components provided by the above goals are able to be used through the CSDMS Component Modeling Tool.

Links:

- Subversion repository: http://csdms.colorado.edu/viewvc/ccafe_gui/trunk/CMT

Goal 6) Educational and Knowledge Transfer Goals

In 2011 we continue work on two overarching EKT goals, firstly to create and test tutorials and a help system for the CSDMS Modeling Tool, and secondly to improve the CSDMS Educational Repository. To advance these two overarching goals in 2011 we: 1) standardized and improved the CMT Help System with detailed descriptions of model equations. We posted our first instructional videos on a newly launched CSDMS YouTube Channel; 2) continued to post model animations, new spreadsheet labs for undergraduate students and more advanced modeling labs in the educational repository.

Accomplishments and Highlights: Every model in the CSDMS Model repository has 5 or more key reference papers listed to make informed model use straightforward. We standardized the look and feel of the Help System of the CMT and improved the CMT Help System with detailed descriptions of model equations for 53 components. No user has to experience CMT components as a black box --- core model equations are only a single-click away. These help pages are by design shared through the CSDMS wiki, which allows the original model developers to improve and intermittently update documentation.

The EKT repository has progressed in presentation and content. We now share our documented educational movies and animations through a YouTube CSDMS science and technology channel, and have received >14,000 views since December 29th, 2010. Real-world earth surface processes movies are collected and brought online with documentation during large earth surface dynamics events, such as the Japan tsunami, March 2011, and Mississippi flooding, May 2011. Quantitative modeling resources for undergraduate teaching are developed as complete sets of student labs, spreadsheet exercises, instructor notes and overarching lesson plans.

Transparency and usability of the CSDMS component modeling tool-CMT

The CSDMS Modeling Tool (CMT) is one of the key products of the CSDMS project; it allows earth scientists with little prior modeling experience to use and couple models for surface dynamics research and education on the CSDMS computing cluster. In 2011 we continued to improve the transparency and usability of the CMT.

Portal and Help System

CMT has it own portal on the wiki website: http://csdms.colorado.edu/wiki/CMT_portal

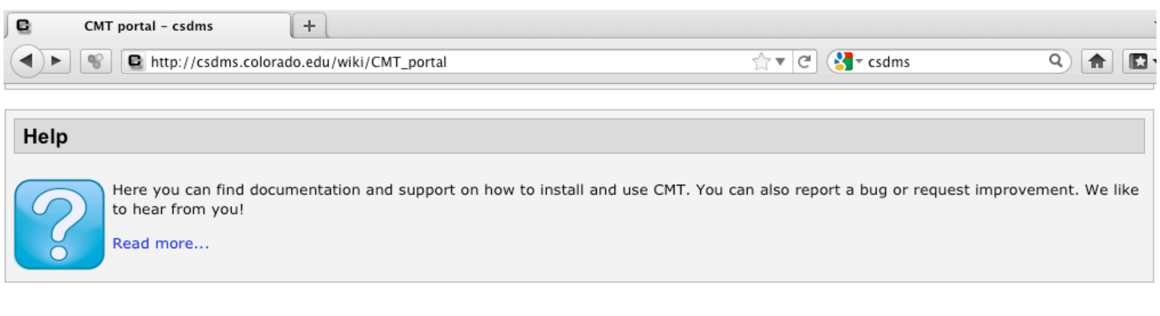
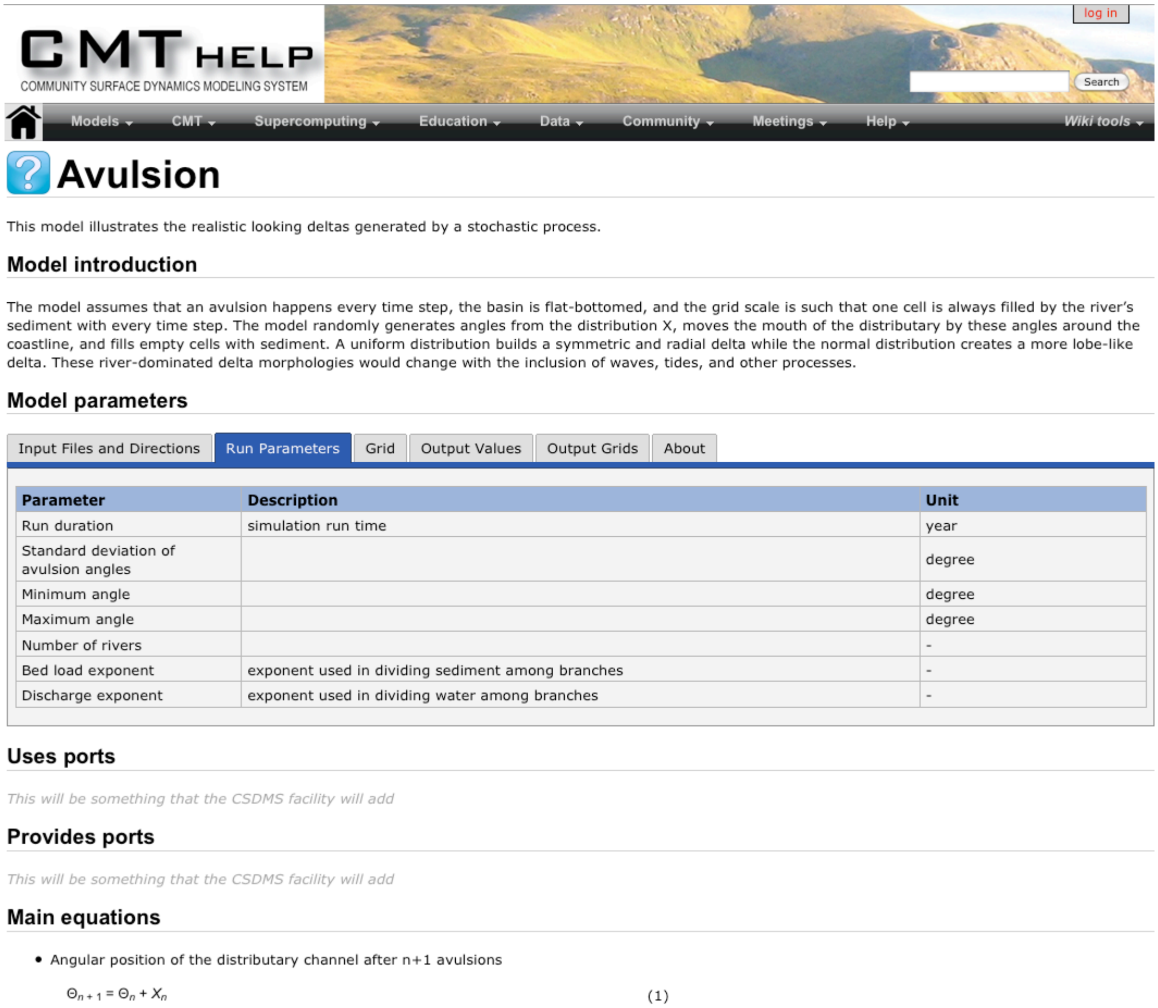


Figure 9. *The CSDMS Component Modeling Tool has a new web portal with a Help System. The Help System refers to navigating and using the CMT, to concise tutorials on starting and running components and to more detailed component help.*

We standardized and further improved the 'CMT Help System' with detailed descriptions of model equations for 53 components. The Help system mirrors tabbed-dialogue user-driven menus in the models themselves. No user has to experience CMT components as a black box, core model equations are only a single-click away for any arbitrary model component. These help pages are intentionally shared through both the CMT directly and through the CSDMS wiki, which allows the original model developers to improve and continuously update documentation.


CMT HELP
COMMUNITY SURFACE DYNAMICS MODELING SYSTEM
log in
Search
Models CMT Supercomputing Education Data Community Meetings Help Wiki tools

Avulsion

This model illustrates the realistic looking deltas generated by a stochastic process.

Model introduction

The model assumes that an avulsion happens every time step, the basin is flat-bottomed, and the grid scale is such that one cell is always filled by the river's sediment with every time step. The model randomly generates angles from the distribution X, moves the mouth of the distributary by these angles around the coastline, and fills empty cells with sediment. A uniform distribution builds a symmetric and radial delta while the normal distribution creates a more lobe-like delta. These river-dominated delta morphologies would change with the inclusion of waves, tides, and other processes.

Model parameters

Input Files and Directions | Run Parameters | Grid | Output Values | Output Grids | About

Parameter	Description	Unit
Run duration	simulation run time	year
Standard deviation of avulsion angles		degree
Minimum angle		degree
Maximum angle		degree
Number of rivers		-
Bed load exponent	exponent used in dividing sediment among branches	-
Discharge exponent	exponent used in dividing water among branches	-

Uses ports

This will be something that the CSDMS facility will add

Provides ports

This will be something that the CSDMS facility will add

Main equations

- Angular position of the distributary channel after n+1 avulsions

$$\Theta_{n+1} = \Theta_n + X_n \quad (1)$$

Figure 10. *Users have single-click access to the model equations behind CMT components. This functionality helps prevent users of experiencing components as a black box --- core model equations are only a single-click away for any arbitrary model component.*

Instructional Videos on CSDMS YouTube channel

We developed a first set of web-based video tutorials that show 1) the vision of the CMT, 2) a beginning user how to install the required software, 3) how to get an account on the supercomputer. More instructional movies will be created and posted in 2011.

Project Governance and Feedback from CMT Users

We value transparency in our CMT software development project. For those CSDMS members that want to monitor progress of development we created a wiki-based progress and workflow-mapping tool. We call this tool a 'component roadmap'; its purpose is to explicitly show what steps a model has to go through before coming online as a CMT component, it also lists the developer or scientist responsible for the steps and sets an approximate timeline.

One more direct feedback option for advanced users is the "Report a bug" option, which allows feedback through the CSDMS Track page. Active tickets are created and posted and are accessible for all stakeholders. Selecting the "Report a bug" option opens a dialog box, in which users may choose whether to create a new ticket for the bug they have discovered, or to view all active tickets.

Figure 11. Instructional videos were launched on the CSDMS YouTube channel; topics include 'How to become a CSDMS member, connecting to the CSDMS HPCC and CMT, Contributing to CSDMS repositories and others. These videos will be expanded in 2011-2012.

Educational Repository 2011

Growing database of documented animations and movies

The EKT repository has further grown to include 93 documented animation and movies. We now share our documented educational movies and animations through a YouTube CSDMS science and technology channel, and have received >14,000 views since December 29th, 2010. Real-world earth surface processes movies are collected and brought online with documentation during large earth surface dynamics events, such as the Japan tsunami, March 2011, and Mississippi flooding, May 2011. This 'rapid response' approach provoked a large number of views: during the May 2011 Mississippi floods the 'CSDMSmovies' YouTube channel had the largest number of views for a not-for-profit science and technology channel.

We intentionally focus on surface dynamics process aspects of these world events. As an example, CSDMS posted a rare movie to explain the concept of a sand boil near a river levee as a result of flood discharge and pressure gradients between the river channel and the surrounding floodplain.

Movies from the educational repository were picked up in early 2011 by the North Carolina Museum of Natural Sciences for video exhibits in their Nature Research Center, as well as by the Oregon Public Broadcasting for their NASA funded educational website on Carbon connections focused on teaching resources on climate science.

Figure 12 CSDMS YouTube movie to explain the concept of a sand boil near a river levee as a result of flood discharge and pressure gradients between the river channel and the surrounding floodplain. Posted during the May 2011 Mississippi floods.

Tiered approach to quantitative modeling: High-school & undergraduate-graduate level teaching resources

The EKT working group proposed to develop the educational repository such that there are different levels of teaching resources on surface process modeling; simple spreadsheet modeling, web-based relatively simple 'slider' models with limited parameter space, and more advanced modeling with CMT.

CSDMS EKT specialist and CSDMS graduate students now have posted a number of spreadsheet exercises with special focus on teaching quantitative skills. The exercises all include student notes, instructor notes, a lesson plan highlighting topical content and which general quantitative skills are being taught. Downloadable labs as of August 1st, 2011 include hydrological processes (e.g. Evaporation, Infiltration and Interception), Delta Evolution (e.g. Sinking Deltas), Glacio-fluvial Processes (e.g. River Discharge Measurements), and a source-to-sink exercise on Sediment Supply and Human Influences.

Outreach Activities

Summer Institute on Earth-Surface Dynamics (NCED/CSDMS)

This two-week institute combines lectures with practical experiences in the laboratory and the field. SIESD' topic in 2011 is 'Coastal Processes and the Dynamics of Deltaic Systems', the course will be held from August 10-19, University of Minnesota. Two days in the summer institute are specially dedicated to use of numerical modeling and quantitative techniques in research and teaching.

A selection of the CMT and spreadsheet exercises will be further tested and evaluated for teaching purposes during this 2-day part of the SIESD course for students, teaching assistants and teaching faculty. This two-week institute combines lectures with practical experiences in the laboratory and the field and now newly expanded with modeling clinics.

Concepts of Supercomputing for Middle School Students

CSDMS scientists and software engineers participated in the INSTAAR Open House 2011. The INSTAAR Open House hosted over 195 middle school students who participated in hands-on science measurements and activities. The CSDMS Integration Facility team set out to teach concepts of super-computing. To illustrate parallel processing, versus fast-processing students raced to perform tasks as 'fast processors' or cluster teams' and gained insights on basic supercomputing strategies. Students played a science game that pitted different computing methods—parallel processors vs. single processors—against each other, using Duplo blocks to perform tasks. Scott Peckham the chief software architect at the Community Surface Dynamics Modeling System conducted the games. "It was interesting—right away the students came up with refinements that mirror stuff we do in programming," Peckham said.

A Component-Based Approach to Integrated Modeling in the Geosciences: The Design of CSDMS

Scott Peckham, Eric Hutton

CSDMS, University of Colorado, 1560 30th Street, UCB 450, Boulder, CO 80309, USA

Boyana Norris

Mathematics and Computer Science Division, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439, USA

Abstract

The development of scientific modeling software increasingly requires the coupling of multiple independently developed models. Component-based software engineering enables the integration of plug-and-play components, but significant additional challenges must be addressed in any specific domain in order to produce a usable development and simulation environment that is also going to encourage contributions and adoption by entire communities. In this paper we describe the challenges in creating a coupling environment for Earth-surface process modeling and how we approach them in our integration efforts at the Community Surface Dynamics Modeling System.

Keywords:
component software, CCA, CSDMS, modeling, code generation

Email addresses: `Scott.Peckham@colorado.edu` (Scott Peckham), `Eric.Hutton@colorado.edu` (Eric Hutton), `norris@mcs.anl.gov` (Boyana Norris)

1. Introduction

The Community Surface Dynamics Modeling System (CSDMS) project [12] is an NSF-funded, international effort to develop a suite of modular numerical models able to simulate a wide variety of Earth-surface processes, on time scales ranging from individual events to many millions of years. CSDMS maintains a large, searchable inventory of contributed models and promotes the sharing, reuse, and integration of open-source modeling software. It has adopted a component-based software development model and has created a suite of tools that make the creation of *plug-and-play* components from stand-alone models as automated and effortless as possible. Models or process modules that have been converted to component form are much more flexible and can be rapidly assembled into new configurations to solve a wider variety of scientific problems. The ease with which one component can be replaced by another also makes it easy to experiment with different approaches to providing a particular type of functionality. The CSDMS project also has a mandate from the NSF to provide a migration pathway for surface dynamics modelers toward high-performance computing (HPC) and provides a 720-core supercomputer for use by its members. In addition, CSDMS provides educational infrastructure related to physically based modeling.

The main purpose of this paper is to present in some detail the key issues and design criteria for a component-based, integrated modeling system and then describe the design choices adopted by the CSDMS project to address these criteria. CSDMS was not developed in isolation: it builds on and extends proven, open-source technology. The CSDMS project also maintains close collaborations with several other integrated modeling projects and seeks to evaluate different approaches in pursuit of those that are optimal. As with any design problem, myriad factors must be considered in determining what is optimal, including how various choices affect users and developers. Other key factors are performance, ease of maintenance, ease of use, flexibility, portability, stability, encapsulation, and future proofing.

1.1. Component Programming Concepts

Component-based programming is all about bringing the advantages of “plug and play” technology into the realm of software. When one buys a new peripheral for a computer, such as a mouse or printer, the goal is to be able to simply plug it into the right kind of port (e.g., a USB, serial, or parallel port) and have it work, right out of the box. For this situation to be possible, however, some kind of published standard is needed that the makers of peripheral devices can design against. For example, most computers have universal serial bus (USB) ports, and the USB standard is well documented. A computer’s USB port can always be expected to provide certain capabilities, such as the ability to transmit data at a particular speed and the ability to provide a 5-volt supply of power with a maximum current of 500 mA. The result of this standardization is that one can usually buy a new device, plug it into a computer’s USB port, and start using it. Software “plug-ins” work in a similar manner, relying on interfaces (ports) that have well-documented structure or capabilities. In software, as in hardware, the term *component* refers to a unit that delivers a particular type of functionality and that can be “plugged in.”

Component programming build on the fundamental concepts of object-oriented programming, with the main difference being the introduction or presence of a runtime *framework*. Components are generally implemented as classes in an object-oriented language, and are essentially "black boxes" that encapsulate some useful bit of functionality.

The purpose of a framework is to provide an environment in which components can be linked together to form applications. The framework provides a number of *services* that are accessible to all components, such as the linking mechanism itself. Often, a framework will also provide a uniform method of trapping or handling exceptions (i.e., errors), keeping in mind that each component will throw exceptions according to the rules of the language that it is written in. In some frameworks (e.g., CCA's Ccaffeine [1]), there is a mechanism by which any component can be promoted to a framework service, as explained in a later section.

One feature that often distinguishes components from ordinary subroutines, software modules, or classes is that they are able to communicate with other components that may be written in a different programming language. This capability is referred to as *language interoperability*. In order for this to be possible, the framework must provide a language interoperability tool that can create the necessary "glue code" between the components. For a CCA-compliant framework, that tool is Babel [14, 29], and the supported languages are C, C++, Fortran (77-2003), Java, and Python. Babel is described in more detail in a later section. For Microsoft's .NET framework [33], that tool is CLR (Common Language Runtime), which is an implementation of an open standard called CLI (Common Language Infrastructure), also developed by Microsoft. Some of the supported languages are C# (a spin-off of Java), Visual Basic, C++/CLI, IronLisp, IronPython, and IronRuby.

CLR runs a form of bytecode called CIL (Common Intermediate Language). Note that CLI does not support Fortran, Java, standard C++, or standard Python.

The Java-based frameworks used by Sun Microsystems are JavaBeans and Enterprise JavaBeans (EJB) [17]. In the words of Armstrong et al. [3]:

> Neither JavaBeans nor EJB directly addresses the issue of language interoperability, and therefore neither is appropriate for the scientific computing environment. Both JavaBeans and EJB assume that all components are written in the Java language. Although the Java Native Interface library supports interoperability with C and C++, using the Java virtual machine to mediate communication between components would incur an intolerable performance penalty on every inter-component function call.

While in recent years the performance of Java codes has improved steadily through just-in-time (JIT) compilation into native code, Java is not yet available on key high-performance platforms such as the IBM Blue Gene/L and Blue Gene/P supercomputers.

Key advantages of component-based programming include the following.

- Components can be written in different languages and still communicate (via language interoperability).
- Components can be replaced, added to, or deleted from an application at runtime via dynamic linking (as precompiled units).

- Components can easily be moved to a remote location (different address space) without recompiling other parts of the application (via RMI/RPC support).
- Components can have multiple different interfaces.
- Components can be "stateful"; that is, data encapsulated in the component is retained between method calls over its lifetime.
- Components can be customized at runtime with configuration parameters.
- Components provide a clear specification of inputs needed from other components in the system.
- Components allow multicasting calls that do not need return values (i.e., send data to multiple components simultaneously).
- Components provide clean separation of functionality (for components, this is mandatory vs. optional).
- Components facilitate code reuse and rapid comparison of different implementations.
- Components facilitate efficient cooperation between groups, each doing what it does best.
- Components promote economy of scale through development of community standards.

2. Background

We briefly overview the component methodology used in CSDMS and the associated tools that support component development and application execution.

2.1. The Common Component Architecture

The Common Component Architecture (CCA) [3] is a *component architecture standard* adopted by federal agencies (largely the Department of Energy and its national laboratories) and academics to allow software components to be combined and integrated for enhanced functionality on high-performance computing systems. The CCA Forum is a grassroots organization that started in 1998 to promote component technology standards (and code reuse) for HPC. CCA defines standards necessary for the interoperation of components developed in different frameworks. Software components that adhere to these standards can be ported with relative ease to another CCA-compliant framework. While a variety of other component architecture standards exist in the commercial sector (e.g., CORBA, COM, .Net, and JavaBeans), CCA was created to fulfill the needs of scientific, high-performance, open-source computing that are unmet by these other standards. For example, scientific software needs full support for complex numbers, dynamically dimensioned multidimensional arrays, Fortran (and other languages), and multiple processor systems. Armstrong et al. [3] explain the motivation for creating CCA by discussing the pros and cons of other component-based frameworks with regard to scientific, high-performance computing. A number of DOE projects, many associated with the Scientific Discovery through Advanced Computing (SciDAC) [46] program, are devoted to the development of component technology for high-performance computing systems. Several of these are heavily invested in the CCA standard (or are moving toward it) and involve computer scientists and applied mathematicians. Examples include the following.

- TASCS: The Center for Technology for Advanced Scientific Computing Software, which focused on CCA and its associated tools [9].
- CASC: Center for Applied Scientific Computing, which is home to CCA's Babel tool [29].
- ITAPS: The Interoperable Technologies for Advanced Petascale Simulation [16], which focuses on meshing and discretization components, formerly TSTT.
- PERI: Performance Engineering Research Institute, which focuses on HPC quality of service and performance issues [30].
- TOPS: Terascale Optimal PDE Solvers, which focuses on PDE solver components [24].
- PETSc: Portable, Extensible Toolkit for Scientific Computation, which focuses on linear and nonlinear PDE solvers for HPC, using MPI [6, 7, 8].

A variety of different frameworks, such as Ccaffeine [1], CCAT/XCAT [25], SciRUN [15] and Decaf [26], adhere to the CCA component architecture standard. A framework can be CCA-compliant and still be tailored to the needs of a particular computing environment. For example, Ccaffeine was designed to support parallel computing, and XCAT was designed to support distributed computing. Decaf [26] was designed by the developers of Babel primarily as a means of studying the technical aspects of the CCA standard itself. The important point is that each of these frameworks adheres to the same standard, thus facilitating reuse of a (CCA) component in another computational setting. The key idea is to isolate the components themselves, as much as possible, from the details of the computational environment in which they are deployed. If this is not done, then we fail to achieve one of the main goals of component programming: code reuse.

CCA has been shown to be interoperable with Earth System Modeling Framework (ESMF) [20] and Model Coupling Toolkit (MCT) [27, 28, 36, 43]. CSDMS has also demonstrated that it is interoperable with a Java version of Open Modeling Interface (OpenMI) [44]. Many of the papers in our cited references have been written by CCA Forum members and are helpful for learning more about CCA. The CCA Forum has also prepared a set of tutorials called "A Hands-On Guide to the Common Component Architecture" [11].

2.2. Language Interoperability with Babel

Babel [29, 14] is an open-source, language interoperability tool (consisting of a compiler and runtime) that automatically generates the "glue code" necessary for components written in different computer languages to communicate. As illustrated in Fig. 1, Babel currently supports C, C++, Fortran (77, 90, 95, and 2003), Java and Python. Babel is much more than a "least common denominator" solution; it even enables passing of variables with

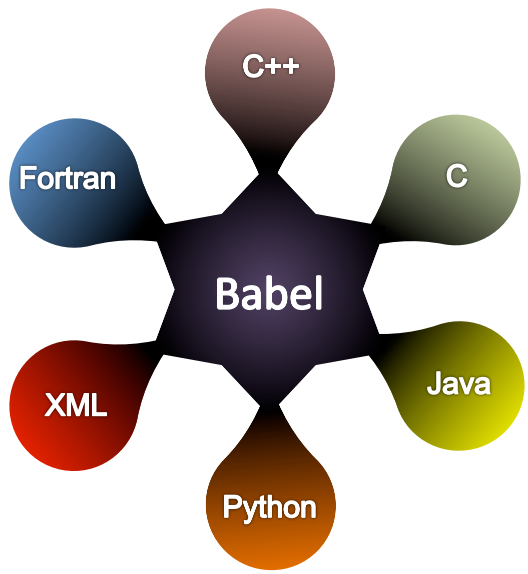


Figure 1: Language interoperability provided by Babel.

data types that may not normally be supported by the target language (e.g., objects and complex numbers). Babel was designed to support *scientific*, *high-performance* computing and is one of the key tools in the CCA tool chain. It won an R&D 100 design award in 2006 for "The world's most rapid communication among many programming languages in a single application." It has been shown to outperform similar technologies such as CORBA and Microsoft's COM and .NET.

In order to create the glue code needed for two components written in different programming languages to exchange information, Babel needs to know only about the interfaces of the two components. It does not need any implementation details. Babel was therefore designed so that it can ingest a description of an interface in either of two fairly "language-neutral" forms, XML (eXtensible Markup Language) or SIDL (Scientific Interface

Definition Language). The SIDL language (somewhat similar to CORBA's IDL) was developed for the Babel project. Its sole purpose is to provide a concise description of a scientific software component interface. This interface description includes complete information about a component's interface, such as the data types of all arguments and return values for each of the component's methods (or member functions). SIDL has a complete set of fundamental data types to support scientific computing, from Booleans to double-precision complex numbers. It also supports more sophisticated data types such as enumerations, strings, objects, structs,and dynamic multidimensional arrays. The syntax of SIDL is similar to that of Java. A complete description of SIDL syntax and grammar can be found in "Appendix B: SIDL Grammar" in the Babel User's Guide [14]. Complete details on how to represent a SIDL interface in XML are given in "Appendix C: Extensible Markup Language (XML)" of the same guide.

2.3. The Ccaffeine Framework

Ccaffeine [1] is the most widely used CCA framework, providing the runtime environment for sequential or parallel components applications. Using Ccaffeine, component-based applications can run on diverse platforms, including laptops, desktops, clusters, and leadership-class supercomputers. Ccaffeine provides some rudimentary MPI communicator services, although individual components are responsible for managing parallelism internally (e.g., communicating data to and from other distributed components). A CCA framework provides *services*, which include component instantiation and destruction, connecting and disconnecting of ports, handling of input parameters, and control of MPI communicators. Ccaffeine was designed primarily to support the single-component multiple-data (SCMD) programming style, although it can support multiple-component multiple-data (MCMD) applications that implement more dynamic management of parallel resources. The CCA specification also includes an event service description, but it is not fully implemented in Ccaffeine yet. Multiple interfaces to configuring and executing component applications within the Ccaffeine framework exist, including a simple scripting language, a graphical user interface, and the ability to take over some of the operations normally handled by the frameworks, such as component instantiation and port connections.

A typical CCA component's execution consists of the following steps:

- The framework loads the dynamic library for the component. Static linking options are also available.
- The component is instantiated. The framework calls the `setServices` method on the component, passing a handle to itself as an argument.
- User-specified connections to other components' ports are established by the framework.
- If the component provides a `gov.cca.ports.Go` port (similar to a "main" subroutine), its `go()` method can be invoked to start the main portion of the computation.
- Connections can be made and broken throughout the life of the component.
- All component ports are disconnected, and the framework calls `releaseServices` prior to calling the component's destructor.

The handle to the framework services object, which all CCA components obtain shortly after instantiation, can be used to access various framework services throughout the component's execution. This represents the main difference between a class and a component: a component dynamically accesses another component's functionality through dynamically connecting ports (requiring the presence of a framework), whereas classes in object-oriented languages call methods directly on instances of other classes.

2.4. Component Development with Bocca

Bocca [2] is a tool in the CCA tool chain that was designed to help users create, edit, and manage a set of SIDL-based entities, including CCA components and ports, that are associated with a particular project. Once a set of CCA-compliant components and ports has been prepared, one can use a CCA-compliant framework such as Ccaffeine to link components from this set together to create applications or composite models.

Bocca was developed to address usability concerns and reduce the development effort required for implementing multilanguage component applications. Bocca was designed specifically to free users from mundane, time-consuming, low-level tasks so they can focus on the scientific aspects of their applications. It can be viewed as a development environment tool that allows application developers to perform rapid component prototyping while maintaining robust software- engineering practices suitable to HPC environments. Bocca provides project management and a comprehensive build environment for creating and managing applications composed of CCA components. Bocca operates in a language-agnostic way by automatically invoking the Babel compiler. A set of Bocca commands required to create a component project can be saved as a shell script, so that the project can be rapidly rebuilt, if necessary. Various aspects of an existing component project can also be modified by typing Bocca commands interactively at a Unix command prompt.

While Bocca automatically generates dynamic libraries, a separate tool can be used to create *stand-alone executables* for projects by automatically bundling all required libraries on a given platform. Examples of using Bocca are available in the set of tutorials called "A Hands-On Guide to the Common Component Architecture," written by the CCA Forum members [11].

2.5. Other Component-Based Modeling Projects

We briefly discuss several other component-based projects in the area of Earth system-related modeling.

- The Object Modeling System (OMS) [35] is a pure Java, object-oriented framework for component-based agro-environmental modeling.

- The Open Modeling Interface (OpenMI) [44] is an open-source software-component *interface standard* for the computational core of numerical models. Model components that comply with this standard can be configured without programming to exchange data during computation (at runtime). Similar to the CCA component model, the OpenMI standard supports two-way links between components so that the involved models can mutually depend on calculation results from each other. Linked models may run asynchronously with respect to time steps, and data represented on different geometries (grids) can be exchanged by using built-in tools for interpolating in space and time. OpenMI was designed primarily for use on PCs, using either the .NET or Java framework. CSDMS has experimented with OpenMI version 1.4 (version 2.0 was recently released) but currently uses a simpler component interface.

- The Earth System Modeling Framework (ESMF) [18, 20] is software for building and coupling weather, climate, and related models written in Fortran. ESMF defines data structures, parallel data redistribution, and other utilities to enable the composition of multimodel high-performance simulations.

- The Framework for Risk Analysis of Multi-Media Environmental Systems (FRAMES) [19] is developed by the U.S. Environmental Protection Agency to provide models and modeling tools (e.g., data retrieval and analysis) for simulating different environmental processes.

3. Problem Definition – Component-based Plug-and-Play Modeling

Next we discuss the challenges that we faced in tackling the problem of creating plug-and-play modeling capabilities that can be extended and actively used by the CSDMS community.

3.1. Attributes of Earth Surface Process Models

The Earth surface process modeling community has *numerous* models, but it is difficult to couple or reconfigure them to solve new problems. The reason is that they are a heterogeneous set.

- The models are written in *many different languages*, which may be object-oriented or procedural, compiled or interpreted, proprietary or open-source, etc. Languages do not all offer the same data types and features, so special tools are required to create "glue code" necessary to make function calls across the *language barrier*.

- The models typically are not designed to "talk" to each other and do not follow any particular set of conventions.
- The models generally have a *geographic* context and are often used in conjunction with GIS (Geographic Information System) tools.
- The generally consist of one or more arrays (1D, 2D, or 3D) that are being advanced in time according to differential equations or other rules (i.e., we are not modeling molecular dynamics).
- The models use different input and output file formats.
- The models are often *open source*. Even many models that were originally sold commercially are now available as open-source code, for example parts of Delt3D from Deltares and many EDF (Energie du Francais) models.

3.2. Difficulties in Linking Models

Linking together models that were not specifically designed from the outset to be linkable is often surprisingly difficult, and a brute-force approach to the problem often requires a significant investment of time and effort. The main reason is that two models may differ in may ways. The following list of possible differences illustrates this point.

- The models are written in different languages, making conversion time-consuming and error-prone.

- The person doing the linking may not be the author of either model, and the code is often not well-documented or easy to understand.
- Models may have different dimensionality (1D, 2D, or 3D).
- Models may use different types of grids (e.g., rectangles, triangles, and Voronoi cells).
- Each model has its own time loop or “clock.”
- The numerical scheme may be either explicit or implicit.

3.3. Design Criteria

The technical goals of a component-based modeling system include the following.

- Support for *multiple operating systems* (especially Linux, Mac OS X, and Windows).
- *Language interoperability* to support code contributions written in procedural languages (e.g., C or Fortran) as well as object-oriented languages (e.g., Java, C++, and Python).
- Support for both *structured and unstructured grids*, requiring a spatial regridding tool.
- *Platform-independent GUIs and graphics* where useful.
- Use of well-established, open-source *software standards* whenever possible (e.g., CCA, SIDL, OGC, MPI, NetCDF, OpenDAP, and XUL).

- Use of *open-source tools* that are mature and have well-established communities, avoiding dependencies on proprietary software whenever possible (e.g., Windows, C#, and Matlab).
- Support for *parallel computation* (multiprocessor, via MPI standard).
- *Interoperability with other coupling frameworks.* Since code reuse is a fundamental tenet of component-based modeling, the effort required to use a component in another framework should be kept to a minimum.
- *Robustness and ease of maintainenance.* It will clearly have many software dependencies, and this software infrastructure will need to be updated on a regular basis.
- Use of *HPC tools and libraries.* If the modeling system runs on HPC architectures, it should strive to use parallel tools and models (e.g., VisIt, PETSc, and the ESMF regridding tool).
- *Familiarity.* Model developers and contributors should not be required to make major changes to how they work.

Expanding the last bullet, developers should not be required to convert to another programming language or use invasive changes to their code (e.g., use specified data structures, libraries, or classes). They should be able to retain "ownership" of the code and make continual improvements to it; someone should be able to componentize future, improved versions with minimal additional effort. The developer will likely want to continue to use the code outside the framework. However, some degree of code refactoring (e.g., breaking code into functions or adding a few new functions) and ensuring that the code compiles with an open-source compiler are considered reasonable requirements. It is also expected that many developers will take advantage of various built-in tools if doing so is straightforward and beneficial.

3.4. Interface vs. Implementation

The word *interface* may be the most overloaded word in computer science. In each case, however, it adheres to the standard, English meaning of the word that has to do with a boundary between two items and what happens at the boundary.

Many people hear the word interface and immediately think of the interface between a human and a computer program, which is typically either a command-line interfaceor a graphical user interface (GUI). While such interfaces are an interesting and complex subject, this is usually not what computer scientists are talking about. Instead, they tend to be interested in other types of interface, such as the one between a pair of software components, or between a component and a framework, or between a developer and a set of utilities (i.e., an API or a software development kit).

Within the present context of component programming, we are interested primarily in the interfaces between components. In this context, the word *interface* has a specific meaning, essentially the same as in the Java programming language. An interface is a user-defined entity/type, similar to an abstract class. It does not have any data fields, but instead is a named set of methods or member functions, each defined completely with regard to argument types and return types but without any actual implementation. A CCA *port* is simply this type of interface. Interfaces are the name of the game when it comes to the question of reusability or “plug and play.” Once an interface has been defined, one can ask the question: Does this component have interface A? To answer the question, we merely have to look at the methods (or member functions) that the component has with regard to their names, argument types, and return types. If a component does have a given interface, then it is said to *expose* or *implement* that interface, meaning that it contains an actual *implementation* for each of those methods. It is fine if the component has additional methods beyond the ones that constitute a particular interface. Thus, it is possible (and frequently useful) for a single component to expose multiple, different interfaces or ports. For example, multiple interfaces may allow a component to be used in a greater variety of settings. An analogy exists in computer hardware, where a computer or peripheral may actually have a number of different ports (e.g., USB, serial, parallel, and ethernet) to enable it to communicate with a wider variety of other components.

The distinction between *interface* and *implementation* is an important theme in computer science. The word pair *declaration* and *definition* is used in a similar way. A function (or class) declaration tells what the function does (and how to interact with or use it) but not how it works. To see how the function actually works, we need to look at how it has been defined or implemented. C and C++ programmers are familiar with this idea, which is similar to declaring variables, functions, classes, and other data types in a header file with the file name extension .h or .hpp, and then defining their implementations in a separate file with extension .c or .cpp.

Of course, most of the gadgets that we use every day (from iPods to cars) are like this. We need to understand their interfaces in order to use them (and interfaces are often standardized across vendors), but often we have no idea what is happening inside or how they actually work, which may be quite complex.

While the tools in the CCA tool chain are powerful and general, they do not provide a ready interface for linking geoscience models (or any domain-specific models). In CCA terminology, *port* is essentially a synonym for interface and a distinction is made between ports that a given component uses (*uses ports*), and those that it provides (*provides ports*) to other components. Note that this model provides a means of bidirectional information exchange between components, unlike dataflow-based approaches (e.g., OpenMI) that support unidirectional links between components (i.e., the data produced by one component is consumed by another component).

Each scientific modeling community that wishes to make use of the CCA tools is responsible for designing or selecting component interfaces (or ports) that are best suited to the kinds of models they wish to link together. This is a big job that involves social as well as technical issues and typically requires a significant time investment. In some disciplines, such as molecular biology or fusion research, the models may look quite different from ours. Ours tend to follow the pattern of a 1D, 2D or 3D array of values (often multiple, coupled arrays) advancing in time. However, our models can still be quite different from each other with regard to their dimensionality or the type of computational grid they use (e.g., rectangles, triangles or polygons), or whether they are implicit or explicit in time.

3.5. Granularity

While components may represent any level of granularity, from a simple function to a complete hydrologic model, the optimum level appears to be that of a particular physical process, such as infiltration, evaporation, or snowmelt. At this level of granularity researchers are most often interested in swapping out one method of modeling a process for another. A simpler method of parameterizing a process may apply only to simplified special cases or may be used simply because there is insufficient input data to drive a more complex model. A different numerical method may solve the same governing equations with greater accuracy, stability, or efficiency and may or may not use multiple processors. Even the same method of modeling a given process may exhibit improved performance when coded in a different programming language. But the physical process level of granularity is also natural for other reasons. Specific physical processes often act within a domain that shares a physically important boundary with other domains (e.g., coastline and ocean-atmosphere), and the fluxes between these domains are often of key interest. In addition, experience shows that this level of granularity corresponds to GUIs and HTML help pages that are more manageable for users.

A judgment call is frequently needed to decide whether a new feature should be provided in a separate component or as a configuration setting in an existing component. For example, a kinematic wave channel-routing component may provide both Manning's formula and the law of the wall as different options to parameterize frictional momentum loss. Each of these options requires its own set of input parameters (e.g., Manning's n or the roughness parameter, z_0). We could even think of frictional momentum loss as a separate physical process, under which we would have a separate Manning's formula and law of the wall components. Usually, the amount of code associated with the option and usability considerations can be used to make these decisions.

Some models are written in such a way that decomposing them into separate process components is not really appropriate, because of some special aspect of the model's design or because decomposition would result in an unacceptable loss of performance (e.g., speed, accuracy, or stability). For example, *multiphysics models*—such as Penn State Integrated Hydrologic Model (PIHM)—represent many physical processes as one large, coupled set of ODEs that are then solved as a matrix problem on a supercomputer. Other models involve several physical processes that operate in the same domain and are relatively tightly coupled within the governing equations. The Regional Ocean Modeling System (ROMS) is an example of such a model, in which it may not be practical to model processes such as tides, currents, passive scalar transport (e.g., T and S), and sediment transport within separate components. In such cases, however, it may still make sense to wrap the entire model as a component so that it may interact with other models (e.g., an atmospheric model, such as WRF, or a wave model, such as SWAN) or be used to drive another model (e.g., a Lagrangian transport model, such as LTRANS).

4. Designing a Modeling Interface

A component interface is simply a named set of functions (called methods) that have been defined completely in terms of their names, arguments and return values. The purpose of this section is to explain the types of functions that are required and why. The functions that define an interface are somewhat analogous to the buttons on a handheld remote control—they provide a caller with fine-grained control of the model component.

4.1. The "IRF" Interface Functions

Most Earth-science models initialize a set of state variables (often as 1D, 2D, or 3D arrays) and then execute of series of timesteps that advance the variables forward in time according to physical laws (e.g., mass conservation) or some other set of rules. Hence, the underlying source code tends to follow a standard pattern that consists of three main parts. The first part consists of all source code prior to the start of the time loop and serves to set up or *initialize* the model. The second part consists of all source code *within* the time loop and is the guts of the model where state variables are updated with each time step. The third part consists of all source code after the end of the time loop and serves to tear down or *finalize* the model. Note that root-finding and relaxation algorithms follow a similar pattern even if the iterations do not represent timestepping. A time-independent model can also be thought of as a time-stepping model with a single time step. For maximum plug-and-play flexibility, each of these three parts must be encapsulated in a separate function that is accessible to a caller. It turns out that we get more flexibility if the function for the middle phase is written to accept the start time and end time as arguments.

For lack of a better term, we refer to this Initialize(), Run_Until(), Finalize() pattern as an ***IRF interface***. All of the model coupling projects that we are aware of use this pattern as part of their component interface, including CSDMS, ESMF, OMF, and OpenMI. An IRF interface is also used as part of the Message Passing Interface (MPI) for communication between processes in high-performance computers.

To see how an IRF interface is used when coupling models, consider two models, Models A and B, that do not have this interface. To combine them into a single model, where one uses the output of the other during its time loop, we would need to cut the code from within Model A's time loop and paste it into Model B, or vice versa. The reason is that both models were designed to control the time loop and cannot reliquish this control.

4.1.1. Initialize (Model Setup)

The initialize step puts a model into a valid state that is ready to be executed. Mostly this involves initializing variables or grids that will be used within the execution step. Temporary files that the execution step will read from or write to should also be opened here.

4.1.2. Run_Until (Model Execution)

The run step advances the model from its current state to a future state. For time-independent models the run step simply executes the model calculation and updates the model state so that future calls will not require executing the calculations again. Encapsulating only the code *within* the time loop allows an application to run the model to intermediate states.

This is necessary to allow an application to query the model's state for the purposes of (for instance) printing output or passing state data to another model.

4.1.3. Finalize (Model Termination)

The finalize step cleans up after the model is no longer needed. The main purpose of this step to make sure that all resources a model acquired through its life have been freed. Most often this will be freeing allocated memory, but it could also be freeing file or network handles. Following this step, the model should be left in an invalid state such that its run step can no longer be called until it has been initialized again.

4.2. Getter and Setter Interface Functions

A basic IRF interface, while important, really provides only the core functionality of a model coupling interface. A complete interface will also require functions that enable another component to request data from the component (a getter) or change data values (a setter) in the component. These are typically called within the Initialize() or Run_Until() methods.

4.2.1. Value Getters

Limiting access to the model's state to be through a set of functions allows control of what data the model shares with other programs and how it shares that data. The data may be transferred in two ways. The first is to give the calling program a copy of the data. The second is to give the actual data that is being used by the model (in C, this would mean passing a pointer to a value). The first has the advantage that it hides implementation details of the model from the calling program and limits what the calling program can do to the model. However, the downside of the first method is that communication will be slower (and could be significantly so, depending on the size of the data being transferred).

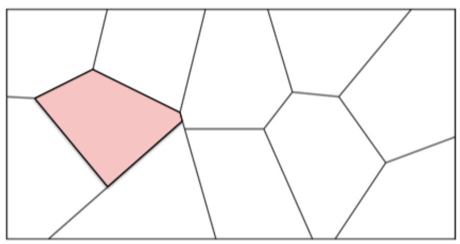
4.2.2. Value Setters

Variables in a model should be accessed and changed only through interface methods. This approach ensures that users of the interface are not able to change values that the interface implementor does not want them to change. This also detaches the programmer using the interface from the model implementation, thus freeing the model developer to change details of the model without an application programmer having to make any changes.

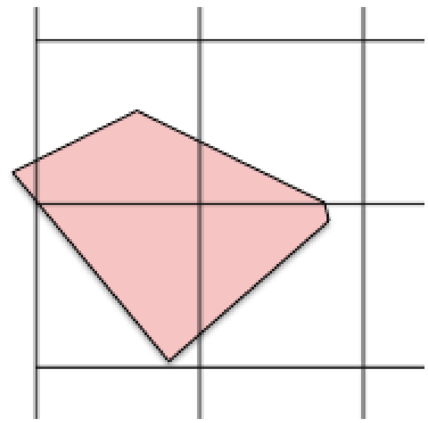
The setter can also perform tasks other than just setting data. For instance, it might be useful if the setter checked to make sure that the new data is valid. After the setter method sets the data, it should ensure that the model is still in a valid state.

The Get_Value() and Set_Value() methods can be general in terms of supporting different grid or mesh types, but it should be possible to bypass that generality and use simple, raster-based grids to keep things simple and efficient when the generality is not needed.

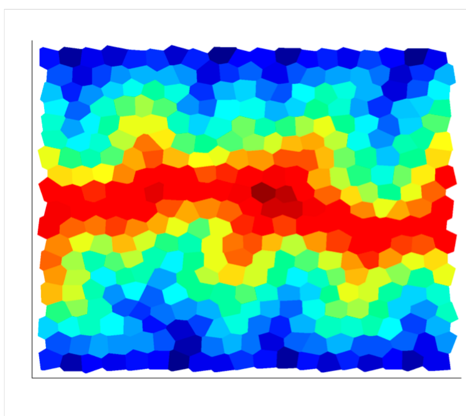
CSDMS has wrapped two open-source regridding tools that can act as services (see Section 9) that other components can use when communicating with one another (an example regridding scenario is shown in Figure 2). The first is from the ESMF project. It is implemented in Fortran and is designed to use multiple processors on a distributed memory system. It supports sophisticated options such as mass-conservative interpolation. The second tool is the multithreaded tool included in the Java SDK for OpenMI.



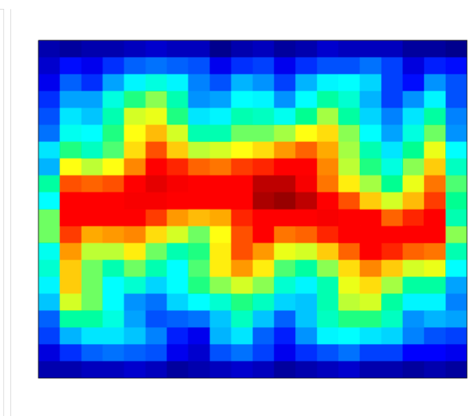
(a) Voronoi cells.



(b) Intersecting raster and Voronoi cells.



(c) Voronoi cells before regridding.



(d) After regridding to raster cells.

Figure 2: Regridding example.

The Get_Value() and Set_Value() methods should optionally allow specification (via indices) of which individual elements within an array that are to be obtained or modified. We often need to manipulate just a few values, and we don'twant to transfer copies of entire arrays (which may be large) unless necessary.

Each component should understand what variables will be requested from it; and if those represent some function of its state variables (e.g., a sum or product), then that computation should be done by the component and offered as an output variable rather than passing several state variables that must then be combined in some way by the caller.

In order to support dynamically typed languages like Python, additional interface functions may be required in order to query whether the variable is currently a scalar or a vector (1D array) or a grid.

4.3. Self-Descriptive Interface Functions

Two additional methods for a modeling interface would enable a caller to query what type of data the component is able to use as input or compute as output. These would typically not require arguments and would simply return the names of all the possible input or output variables as an array of strings, for example Get_Input_Item_List() and Get_Output_Item_List(). Another type of self-descriptive function would be a function like Get_Status() that returns the component's current status as a string from a standardized list.

4.4. Framework Interface Functions

A component typically needs some additional methods that allow it to be instantiated by and communicate with a component-coupling framework. For example, a component must implement methods called __init__(), getServices(), and releaseServices() in order to be used within a CCA-compliant framework.

4.5. Autoconnection Problem

A key goal of component-based modeling is to create a collection of components that can be coupled together to create new and useful composite models. This goal can be achieved by providing every component with the same interface, and this is the approach used by OpenMI. A secondary goal, however, is for the coupling process to be as automatic as possible, that is, to require as little input as possible from users. To achieve this goal, we need some way to group components into categories according to the functionality they provide. This grouping must be readily apparent to both a user and the framework (or system) so that it is clear whether a particular pair of components are *interchangeable*. But what should it mean for two components to be interchangeable? Do they really need to use identical input variables and provide identical output variables? Our experience shows that this definition of interchangeable is unnecessarily strict.

To bring these issues into sharper focus, consider the physical process of infiltration, which plays a key role in hydrologic models. As part of a larger hydrologic model, the main purpose of an infiltration component is to compute the infiltration rate at the surface, because it represents a loss term in the overall hydrologic budget. If the domain of the infiltration component is restricted to the unsaturated zone, above the water table, then it may also need to provide a vertical flow rate at the water table boundary. Thus, the main job of the infiltration component is to provide fluxes at the (top and bottom) boundaries of its domain. To do this job, it needs variables such as flow depth and rainfall rate that are outside its domain and computed by another component. Hydrologists use a variety of different methods and approximations to compute surface infiltration rate. The Richards 3D method, for example, is a more rigorous approach that tracks four state variables throughout the domain; on the other hand, the Green-Ampt method makes a number of simplifying assumptions so that it computes a smaller set of state variables and does not resolve the vertical flow dynamics to the same level of detail (i.e., piston flow, sharp wetting front). As a result, the Richards 3D and Green-Ampt infiltration components use a different set of input variables and provide a different set of output variables. Nevertheless, they both provide the surface infiltration rate as one of their outputs and can therefore be used "interchangeably" in a hydrologic model as an "infiltration component."

The infiltration example illustrates several key points that are transferable to other situations. Often a model, such as a hydrologic model, breaks the larger problem domain into a set of subdomains where one or more processes are relevant. The boundaries of these subdomains are often physical interfaces, such as surface/subsurface, unsaturated/saturated zone, atmosphere/ocean, ocean/seafloor, or land/water. Moreover, the variables that are of interest in the larger model often depend on the fluxes across these subdomain boundaries.

Within a group of interchangeable components (e.g., infiltration components), there are many other implementation differences that a modeler may wish to explore, beyond just how a physical process is parameterized. For example, performance and accuracy often depend on the numerical scheme (explicit vs. implicit, order of accuracy, stability), data types used (float vs. double), number of processors (parallel vs. serial), approximations used, the programming language, or coding errors.

Autoconnection of components is important from a user's point of view. Components typically require many input variables and produce many output variables. Users quickly become frustrated when they need to manually create all these pairings/connections, especially when using more than just two or three components at a time. The OpenMI project does not support the concept of auto-connection or interchangeable components. When using the graphical Configuration Editor provided in its SDK, users are presented with droplists of input and output variables and must select the ones to be paired. Doing so requires expertise and is made more difficult because there is so far no ontological or semantic scheme to clarify whether two variable names refer to the same item.

The CSDMS project currently employs an approach to autoconnection that involves providing interfaces (i.e. ,CCA ports) with different names to reflect their intended use (or interchangeability), even though the interfaces are the same internally.

5. Current CSDMS Component Interface

This section contains a concise list of the current CSDMS IRF and getter/setter interfaces, which must be implemented by any compliant components.

5.1. The IRF Interface

The following methods comprise the IRF interface described in more detail in Section 4.1.

CMI_INITIALIZE (handle, filename)

OUT	handle	handle to the CMI object
IN	filename	path to configuration file

CMI_RUN_UNTIL (handle, stop_time)

IN	handle	handle to the CMI object
IN	stop_time	simulation time to run model until

CMI_FINALIZE (handle)

INOUT	handle	handle to the CMI object

5.2. Value Getters and Setters

The following methods comprise the CSDMS getter/setter interface discussed in Section 4.2.

CMI_GRID_DIMEN (handle, value_str, dimen)

IN	handle	handle to the CMI object
IN	value_str	name of the value to get
OUT	dimen	length of each grid dimension

CMI_GRID_RES (handle, value_str, res)

IN	handle	handle to the CMI object
IN	value_str	name of the value to get
OUT	res	grid spacing for each dimension

CMI_GET_GRID_DOUBLE (handle, value_str, buffer)

IN	handle	handle to the CMI object
IN	value_str	name of the value to get
OUT	buffer	initial address of the destination values

CMI_SET_GRID_DOUBLE (handle, value_str, buffer, dimen)

IN	handle	handle to the CMI object
IN	value_str	name of the value to get
IN	buffer	initial address of the source values
IN	dimen	grid dimension

CMI_GET_TIME_SPAN (handle, span)

IN	handle	handle to the CMI object
OUT	span	start and end times for the simulation

CMI_GET_ELEMENT_SET (handle, value_str, element_set)

IN	handle	handle to the CMI object
IN	value_str	name of the value to get
OUT	buffer	model ElementSet

CMI_GET_VALUE_SET (handle, value_str, value_set)

IN	handle	handle to the CMI object
IN	value_str	name of the value to get
OUT	buffer	model ValueSet

CMI_SET_VALUE_SET (handle, value_str, value_set)

IN	handle	handle to the CMI object
IN	value_str	name of the value to get
IN	buffer	model ValueSet

6. Component Wrapping Issues

In this section we discuss several methods for creating components based on existing codes by using an approach often referred to as *wrapping*.

6.1. Code Reuse and the Case for Wrapping

Using computer models to simulate, predict, and understand Earth surface processes is not a new idea. Many models exist, some of which are fairly sophisticated, comprehensive, and well tested. The difficulty with reusing these models in new contexts or linking them to other models typically has less to do with how they are implemented and more to do with the interface through which they are called (and to some extent, the implementation language.) For a small or simple model, little effort may be needed to rewrite the model in a preferred language and with a particular interface. Rewriting large models, however, is both time-consuming and error prone. In addition, most large models are under continual development, and a rewritten version will not see the benefits of future improvements. Thus, for code reuse to be practical, we need a *language interoperability tool*, so that components dont need to be converted to a different language, and a *wrapping procedure* that allows us to provide existing code with a new calling interface. As suggested by its name, and the fact that it applies to the "outside" (interface) of a component vs. its "inside" (implementation), wrapping tends to be noninvasive and is a practical way to convert existing models into components.

6.2. Wrapping for Object-Oriented Languages

Component-based programming is essentially object-oriented programming with the addition of a framework. If a model has been written as a class, then it is relatively straightforward to modify the definition of this class so that it exposes a particular model-coupling interface. Specifically, one could add new methods (member functions) that call existing methods, or one could modify the existing methods. Each function in the interface has access to all of the state variables (data members) without passing them explicitly; it also has access to all the other interface functions. In object-oriented languages one commonly distinguishes between *private methods* that are intended for internal use by the model object and *public methods* that are to be used by callers and that may comprise one or more interfaces. (Some languages, like Java, make this part of a method's declaration.)

In order for this model object to be used as a component in a CCA-compliant framework like Ccaffeine, it must also be "wrapped" by a CCA implementation file (or IMPL file). The CCA tool chain has tools such as Babel and Bocca that are used to autogenerate an IMPL-file template. For a model that is written in an object-oriented and Babel-supported language (e.g., C++, Python, or Java), the IMPL file needs to do little more than add interface functions like setServices and releaseServices that allow the component to communicate with and be instantiated by the framework. The interface functions used for intercomponent communication (i.e., passing data and IRF) can simply be inherited from the model class. Inheritance is a standard mechanism in object-oriented languages that allows one interface (set of methods) to be extended or overridden by another. Note that the IMPL file may have its own Initialize() function that first gets the required CCA ports and then calls the Initialize() function in the model's interface. But the function that gets the CCA ports can simply be another function in the model's interface that is used only in this context. Similarly, the IMPL file may have a Finalize() function that calls the Finalize() function of the model and then calls a function to release the CCA ports that are no longer needed. It is desirable to keep the IMPL files as clean as possible, which means adding some CCA-specific functions to the model's interface. For example, a CSDMS component would have (1) functions to get and release the required CCA ports, (2) a function to create a tabbed-dialog (using CCA's so-called parameter ports), and (3) a function that prints a language-specific traceback to stdout if an exception occurs during a model run.

6.3. Wrapping for Procedural Languages

Languages such as C or Fortran (up to 2003) do not provide object-oriented primitives for encapsulating data and functionality. Because component-based programming requires such encapsulation, the CCA provides a means to produce object-oriented software even in languages that do not support it directly. We briefly describe the mechanism for creating components based on functionality implemented in a procedural language (e.g., an existing library or model).

A class in object-oriented terminology encapsulates some set of related functions and associated data. To wrap a set of library functions, one can create a SIDL interface or class that contains a set of methods whose implementations call the legacy functions. The new interface does not have to mirror existing functions exactly, presenting a nonintrusive opportunity for redesigning the publicly accessible interfaces presented by legacy software. The creation of class or component wrappers also enables the careful definition of namespaces, thus reducing potential conflicts when integrating with other classes or components. The SIDL definitions are processed by Babel to generate IMPL files in the language of the code being wrapped. The calls to the legacy library can then be added either manually or by a tool, depending on how closely the SIDL interface follows the original library interface.

Function argument types that appear in the SIDL definition can be handled in two ways: by using a SIDL type or by specifying them as *opaque*. SIDL already supports most basic types and different kinds of arrays found in the target languages. Any user-defined types (e.g., structs in C or derived types in Fortran) must have SIDL definitions or be passed as opaques. Because opaques are not accessible from components implemented in a different language, they are rarely used. Model state variables that must be shared among components can be handled in a couple of ways. They can be encapsulated in a SIDL class and accessed through get/set methods (e.g., as described in Section 4.2). Recently Babel has added support for defining *structs* in SIDL, whose data members can be accessed directly from multiple languages.

SIDL supports namespacing of symbols through the definition of packages whose syntax and semantics are similar to Java's packages. In languages that do not support object orientation natively, symbols (e.g., function names) are prefixed with the names of all enclosing packages and parent class. This approach greatly reduces the potential build-, link-, or runtime name conflicts that can result when multiple components define the same interfaces (e.g., the initialize, run, and finalize methods). These naming conventions can be applied to any code, not only SIDL-based components.

Implementors working in non object-oriented languages should encapsulate their model's state data in an object that is opaque to the application programmer. Memory within the object is not directly accessible by the user but can be accessed through an opaque handle, which exists in user space. This handle is passed as the first argument to each of the interface functions so that they can operate on a particular instance of a model. For example, in C, this handle could simply be a pointer to the object and in Fortran, the handle could be an index into a table of opaque objects in a system table.

Model handles are allocated and deallaocated in the initialize and finalize interface functions, respectively. For allocate calls, the initialize functions are passed an OUT argument that will contain a valid reference to the object. For deallocation, the finalize function accepts an INOUT variable that provides a reference to the object to be destroyed and sets the object to an invalid state.

6.4. Guidelines for Model Developers

Developers can follow several relatively simple follow so that it becomes much easier to create a reusable, plug-and-play component from their model source code. Given the large number of models that are contributed to the CSDMS project, it is much more efficient for model developers to follow these guidelines and thereby "meet us halfway" than for CSDMS staff to make these changes after code has been contributed. This can be thought of as a form of load balancing.

6.4.1. Programming Language and License

- Write code in a Babel-supported language (C, C++, Fortran, Java, Python).

- If code is in Matlab or IDL, use tools like I2PY to convert it to Python. Python (with the numpy, scipy, and matplotlib packages) provides a free work-alike to Matlab with similar performance.

- Make sure that code can be compiled with an open-source compiler (e.g., gcc and gfortran).

- Specify what type of open-source license applies to your code. Rosen [41] provides a good, online, and open-source book that explains open-source licensing in detail. CSDMS requires that contributions have an open source license type that is compliant with the standard set forth by the Open Source Initiative.

6.4.2. Model Interface

- Refactor the code to have the basic IRF interface (5.1).

- If code is in C or Fortran, add a model name prefix to all interface functions to establish a namespace (e.g., ROMS_Initialize()). C code can alternatively be compiled as C++.

- Write Initialize() and Run_Until() functions that will work whether the component is used as a driver or *nondriver*.

- Provide getter and setter functions (4.2.1).

- Provide functions that describe input and output *exchange items* (4.2.1).

- Use descriptive function names (e.g., Update_This_Variable).

- Remove user interfaces, whether graphical, command line or otherwise, from your interface implementation. This avoids incompatible user interfaces competing with one another.

6.4.3. State Variables

- Decide on an appropriate set of state variables to be maintained by the component and made available to callers.

- Attempt to minimize data transfer between components (as discussed above).

- Use descriptive variable names.

- Carefully track each variable's units.

6.4.4. Input and Output Files

- Do not hardwire configuration settings in the code; read them from a configuration file (text).

- Do not use hardwired input filenames.

- Read configuration settings from text files (often in Initialize()). Do not prompt for command-line input. If a model has a GUI, write code so it can be bypassed; use the GUI to create a configuration file.

- Design code to allow separate input and output directories that are read from the configuration file. This approach allows many users to use the same input data without making copies (e.g., test cases). It is frequently helpful to include a *case prefix* (scenario) and a *site prefix* (geographic name) and use them to construct default output filenames.

- Establish a namespace for configuration files (e.g., ROMS_input.txt vs. input.txt).

- If large arrays are to be stored in files, save them as binary vs. text. (e.g., this is the case with NetCDF)

- Provide self-test functions or unit tests and test data. One self-test could simply be a "sanity check" that uses trivial (perhaps hard-coded) input data. When analytic solutions are available, these make excellent self-tests because they can also be used to check the accuracy and stability of the numerical methods.

- Do not create and write to output files within the interface implementation. If this is not possible, output files should be well documented and allow for a naming convention that reduces the possibility of naming conflicts.

6.4.5. Documentation

- Help CSDMS to provide a standardized, HTML help page.

- Help CSDMS to provide a standaridized, tabbed-dialog GUI.

- Make liberal use of comments in the code.

7. The CSDMS Modeling Tool (CMT)

As explained in Section 2.3, Ccaffeine is a CCA-compliant framework for connecting components to create applications. From a user's point of view, Ccaffeine is a low-level tool that executes a sequence of commands in a Ccaffeine script. The (natural language) commands in the Ccaffeine scripting language are fairly straightforward, so it is not difficult for a programmer to write one of these scripts. For many people, however, using a graphical user interface (GUI) is preferable because they don'thave to learn the syntax of the scripting language. A GUI also provides users with a natural, visual representation of the connected components as boxes with buttons connected by wires. It can also prevent common scripting errors and offer a variety of other convenient features. The CCA Forum developed such a GUI, called Ccafe-GUI, that presented components as boxes in a palette that can be moved into an arena (workspace) and connected by wires. It also allows component configurations and settings to be saved in BLD files and instantly reloaded later. Another key feature of this GUI is that, as a lightweight and platform-independent tool written in Java, it can be installed and used on any computer with Java support to create a Ccaffeine script. This script can then be sent to a remote, possibly high-performance computer for execution.

While the Ccafe-GUI was certainly useful, the CSDMS project realized that it could be improved and extended in numerous ways to make it more powerful and more user-friendly. In addition, these changes would serve not only the CSDMS community but could be shared back with the CCA community. That is, the new GUI works with any CCA-compliant components, not just CSDMS components. The new version is called CMT (CSDMS Modeling Tool). Significant new features of CMT 1.5 include the following.

- Integration with a powerful visualization tool called VisIt (see below).
- New, "wireless" paradigm for connecting components (see below).

- A login dialog that prompts users for remote server login information.
- Job management tools that are able to submit jobs to processors of a cluster.
- "Launch and go": launch a model run on a remote server and then shut down the GUI (the model continues running remotely).
- New File menu entry: "Import Example Configuration."
- A Help menu with numerous help documents and links to websites.
- Ability to submit bug reports to CSDMS.
- Ability to do file transfers to and from a remote server.
- Help button in tabbed dialogs to launch component-specific HTML help.
- Support for droplists and mouse-over help in tabbed dialogs.
- Support for custom project lists (e.g., projects not yet ready for release).
- A separate "driver palette" above the component palette.
- Support for numerous user preferences, many relating to appearance.
- Extensive cross-platform testing and "bulletproofing."

The CMT provides integrated visualization by using VisIt. VisIt [47] is an open-source, interactive, parallel visualization and graphical analysis tool for viewing scientific data. It was developed by the U.S. Department of Energy Advanced Simulation and Computing Initiative to visualize and analyze the results of simulations ranging from kilobytes to terabytes. VisIt was designed so that users can install a client version on their PC that works together with a server version installed on a high-performance computer or cluster. The server version uses multiple processors to speed rendering of large data sets and then sends graphical output back to the client version. VisIt supports about five dozen file formats and provides a rich set of visualization features, including the ability to make movies from time-varying databases. The CMT provides help on using VisIt in its Help menu. CSDMS uses a service component to provide other components with the ability to write their output to NetCDF files that can be visualized with VisIt. Output can be 0D, 1D, 2D, or 3D data evolving in time, such as a time series (e.g., a hydrograph), a *profile series* (e.g., a soil moisture profile), a 2D *grid stack* (e.g., water depth), a 3D *cube stack*, or a scatter plot of XYZ triples.

Another innovative feature of CMT 1.5 is that it allows users to toggle between the original, *wired* mode and a new *wireless* mode. CSDMS found that displaying connections between components with the use of wires (i.e., red lines) did not scale well to configurations that contained several components with multiple ports. In wireless mode, a component that is dragged from the palette to the arena appears to broadcast what it can provide (i.e., CCA provides ports) to the other components in the arena (using a concentric circle animation). Any components in the arena that need to use that kind of port get automatically linked to the new one; this is indicated through the use of unique, matching colors. In cases where two components in the arena have the same *uses port* but need to be connected to different providers, wires can still be used.

CSDMS continues to make usability improvements to the CMT and used the tool to teach a graduate-level course on surface process modeling at the University of Colorado, Boulder, in 2010. Several features of the CMT make it ideal for teaching, including (1) the ability to save prebuilt component configurations and their settings in BLD files, (2) the File >> Import Example Configuration feature, (3) a standardized HTML help page for each component, (4) a uniform, tabbed-dialog GUI for each component, (5) rapid comparison of different approaches by swapping one component for another, (6) the simple installation procedure, and (7) the ability to use remote resources.

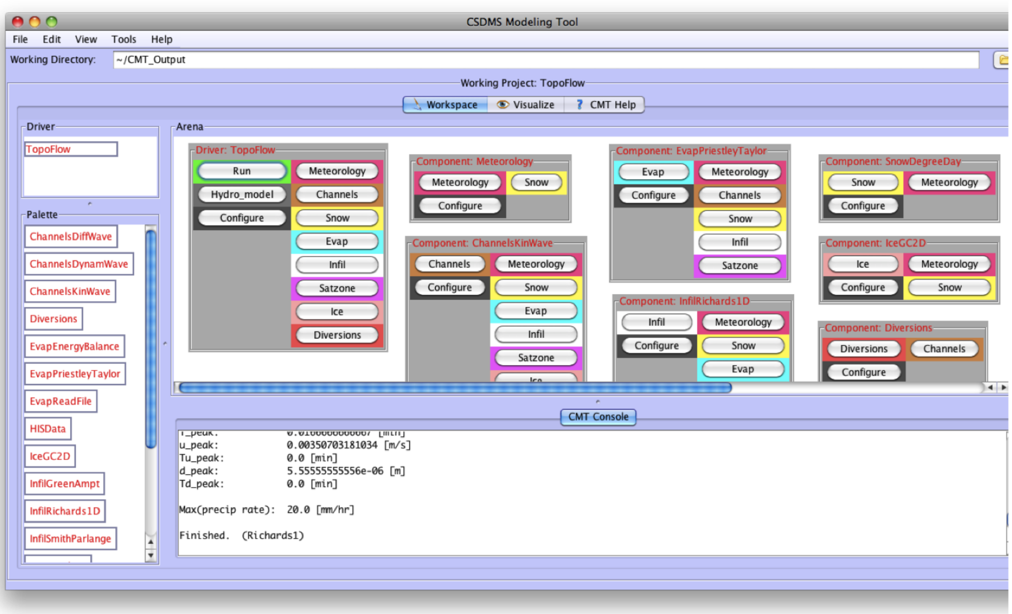


Figure 3: CMT screenshot.

8. Providing Components with a Uniform Help System and GUI

Beyond the usual software engineering definition of a component, a useful component will be one that also comes bundled with metadata that describes the component and the underlying model that it is built around. While creating a component as described in the preceding sections is important, it is of equal importance to have a well-documented component that an end user is able to easily use.

With a plug-and-play framework where users easily connect, interchange, and run coupled models, there is a tendency for a user to treat components as black boxes and ignore the details of the foundation that each component was built upon. For instance, if a user is unaware of the assumptions that underlie a model, that user may couple two components for which coupling does not make sense because of the physics of each model. The user may attempt to use a component in a situation where it was not intended to be used. To combat this problem, components are bundled with HTML help documents, which are easily accessible through the CMT, and describe the component and the model that it wraps. These documents include the following.

- Extended model description (along with references)
- Listing and brief description of the component's uses and provides ports
- Main equations of the model
- Sample input and output
- Acknowledgment of the model developer(s)

A complete component also comes with metadata supplied in a more structured format. Components include XML description files that describe their user-editable input variables. These description files contain a series of XML elements that contain detailed information about each variable including a default value, range of acceptable values, short and long descriptions, units, and data type.

```
<entry name=velocity>
 <label>River velocity</label>
 <help>Depth-averaged velocity at the river mouth</help>
 <default>2</default>
 <type>Float</type>
 <range>
   <min>0</min>
   <max>5</max>
 </range>
 <units>m/s</units>
</entry>
```

Using this XML description, the CMT automatically generates a graphical user interface (in the form of tabbed dialogs) for each CSDMS component. Despite each model's input files being significantly different, this provides CMT users with a uniform interface across all components. Furthermore, the GUI checks user input for errors and provides easily accessible help within the same environment—none of which is available in the batch interface of most models. A special type of CCA *provides port* called a *parameter port* is also used in the creation of the tabbed dialogs.

Nearly every model gathers initial settings from an input file and then runs without user intervention. Ultimately, any user interface that wraps a model must generate this input file for the component to read as part of its initialization step. The above XML description along with a template input file allows this to happen. Once input is gathered from the user, a model-specific input file is created based on a template input file provided with each component. A valid input file is created based on `$`-based substitutions in this template file. Instead of actual values, the template file contains substitution placeholders of the form `$identifier`. Each identifier corresponds to an entry name in the XML description file and, upon substitution, is replaced by the value gathered from an external user interface (the CMT GUI, for instance).

9. Framework Services: "Built-in" Tools That Any Component Can Use

Developers (e.g., CSDMS staff) may wish to make certain low-level tools or utilities available so that any component (or component developer) can use them without requiring any action from a user. These tools can be encapsulated in special components called *service components* that are automatically instantiated by a CCA framework on startup. The services or methods provided by these components are then called *framework services*. Unlike other components, which users may assemble graphically into larger applications, users do not interact with service components directly. However, a component developer can make calls to the methods of service components through *service ports*. The use of service components allows developers to maintain code for a shared functionality in a single place and to make that functionality available to all components regardless of the language they are written in (or which address space they are in). CSDMS uses service components for tasks such as (1) providing component output variables in a form needed by another component (e.g., spatial regridding, interpolation in time, and unit conversion) and (2) writing component output to a standard format such as NetCDF.

Any CCA component can be "promoted" to a service component. A developer simply needs to add lines to its setServices() method that register it as a framework service. CCA provides a special port for this, *gov.cca.ports.ServiceRegistry*, with three methods: addService(), addSingletonService(), and removeService(). If a developer then wants another component to be able to use this framework service, a call to the gov.cca.Services.getPort() method must be added within its setServices() method. (A similar call must be added in order to use CCA parameter ports and ports provided by other types of components.) Note that the setServices() method is defined as part of the gov.cca.Component interface.

CCA components are designed for use within a CCA-compliant framework (like Ccaffeine) and may make use of service components. But what if we want to use these components outside of a CCA framework? One option is to encapsulate a set of functionality (e.g., a service component) in a SIDL class and then "promote" this class to (SIDL) component status through inheritance and by adding only framework-specific methods like setServices(). (Note that a CCA framework is the entity that calls a component's setServices() method as described in Section 2.3.) This approach can be used to provide both component and noncomponent versions of the class. Compiling the noncomponent version in a Bocca project generates a library file that we can link against or, in the case of Python, a module that we can import.

10. Current Contents of the CSDMS Component Repository

At the time of this publication the CSDMS model repository contains more than 160 models and tools. Of those, 50 have been converted into components as described in this paper and can be used in coupled modeling scenarios with the CMT or through the component composition interfaces supported by Ccaffeine. An up-to-date list is maintained at the CSDMS webiste. As with the model repository as a whole, CSDMS components cover the breadth of surface dynamics systems. Hydrologic components cover various scales ranging from basin-scale (the entire TopoFlow [39] suite of models consists of 15 components that cover infiltration, meteorology, and channel dynamics; HydroTrend [4, 23]) to reach-scale (the one-dimensional sediment transport models of Parker [38]). Terrestrial components include models of landscape evolution (Erode, and CHILD [45]), geodynamics (Subside [21]) and cryospherics (GC2D [22]). Coastal and marine models include Ashton-Murray Coastal Evolution Model [4, 5], Avulsion [4], and the stratigraphic model sedflux [21]. The component repository also contains modeling tools such as the ESMF and OpenMI SDK grid mappers, and file readers and writers for standard file formats (NetCDF, VTK, for example).

11. Conclusions

CSDMS uses a component-based approach to integrated modeling and draws on the combined power of many different open-source tools such as Babel, Bocca, Ccaffeine, the ESMF regridding tool, and the VisIt visualization tool. CSDMS also draws on the combined knowledge and creative effort of a large community of Earth-surface dynamics modelers and computer scientists. Using a variety of tools, standards, and protocols, CSDMS converts a heterogeneous set of open-source, user-contributed models into a suite of plug-and-play modeling components that can be reused in many different contexts. Components that encapsulate a physical process usually represent an optimal level of granularity. Standards that CSDMS has adopted and promotes include CCA, NetCDF [34], HTML, OGC (Open Geospatial Consortium) [37], MPI (Message Passing Interface) [32] and XML [48].

All the software that underlies CSDMS is installed and maintained on its high-performance cluster. CSDMS members have accounts on this cluster and access its resources using a lightweight, Java-based client application called the CSDMS Modeling Tool (CMT) that runs on virtually any desktop or laptop computer. This approach can be thought of as a type of *community cloud* since it provides remote access to numerous resources. This centralized cloud approach offers many advantages including (1) simplified maintenance, (2) more reliable performance, (3) automated backups, (4) remote storage and computation (user's PC remains free), (5) ability for many components (such as ROMS) and tools (such as VisIt and ESMF's regridder) to use parallel computation, (6) requiring to install only a lightweight client on their PC, (7) little technical support needed by users, and (8) ability to submit and run multiple jobs.

Babel's support of the Python language has proven very useful. Python is a modern, open-source, object-oriented language with source code that is easy to write, read and maintain. It runs on virtually any platform. It is useful for system administration, model integration, rapid prototyping, high-level tool development, visualization (via the matplotlib package) and numerical modeling (via the numpy package). Bocca is written in Python, the VisIt visualization package has a powerful Python API, and ESRI's ArcGIS software now uses Python as its scripting language ([10]). Many third-party geographic information system (GIS) tools implemented in Python are also available. With the numpy, scipy, and matplotlib packages, Python provides a work-alike to commercial languages like Matlab with similar performance. Other Python packages that CSDMS has found useful are *suds* (for SOAP-based web services) and *PyNIO* (an API for working with NetCDF files).

Several exciting opportunities exist for further streamlining and expanding the capabilities of CSDMS. One area of particular interest is how CSDMS can provide its members with multiple paths to parallel computation. Software may be designed from the outset to use multiple processors, or be refactored to do so, often using MPI or OpenMP. But this is not easy and typically requires a multiyear investment. Another way to harness the power of parallelism is to modify code to take advantage of numerical toolkits such as PETSc (Portable Extensible Toolkit for Scientific Computation) [6, 7, 8] that contain parallel solvers for many of the differential equations that are used in physically based models. A third way is to for models written in array-based languages such as IDL, Matlab [31] and Python/NumPy [42] to use array-based functions and operators that have been parallelized. This approach, although available only in commercial packages at present, is attractive for several reasons: (1) developers in these languages already know to avoid spatial loops and use the array-based functions whenever possible for good performance, (2) most of these array-based functions are straightforward to parallelize, and (3) developers need only import a different package to take advantage of the parallelized functions.

Web services provide many additional opportunities. Peckham and Goodall [40] have demonstrated how CSDMS components can use CUAHSI-HIS [13] web services to retrieve hydrologic data, but CSDMS components could also offer their capabilities as web services.

CSDMS is also interested in *automated component wrapping*, which can be achieved by adding special annotation keywords within comments in the source code. If the code is sufficiently annotated, it is possible to write a flexible tool to wrap the component with any desired interface. Unfortunately, most existing code has not been annotated in this way, and it is typically necessary to involve the code's developer in the annotation process.

Acknowledgments

CSDMS gratefully acknowledges major funding through a cooperative agreement with the National Science Foundation (EAR 0621695). Additional work was supported by the Office of Advanced Scientific Computing Research, Office of Science, U.S. Dept. of Energy, under Contracts DE-AC02-06CH11357 and DE-FC-0206-ER-25774.

References

[1] Allan, B., Armstrong, R., Lefantzi, S., Ray, J., Walsh, E., Wolfe, P., 2010. Ccaffeine – a CCA component framework for parallel computing. http://www.cca-forum.org/ccafe/.

[2] Allan, B. A., Norris, B., Elwasif, W. R., Armstrong, R. C., Dec. 2008. Managing scientific software complexity with Bocca and CCA. Scientific Programming 16 (4), 315–327.

[3] Armstrong, R., Gannon, D., Geist, A., Keahey, K., Kohn, S., McInnes, L., Parker, S., Smolinski, B., 1999. Toward a Common Component Architecture for high-performance scientific computing. In: Proc. 8th IEEE Int. Symp. on High Performance Distributed Computing.

[4] Ashton, A., Kettner, A. J., Hutton, E. W. H., 2011. Progress in coupling between coastline and fluvial dynamics. Computers & Geosciences (this issue).

[5] Ashton, A., Murray, A. B., Arnoult, O., 2001. Formation of coastline features by large-scale instabilities induced by high-angle waves. Nature 414, 296–300.

[6] Balay, S., Brown, J., , Buschelman, K., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., Zhang, H., 2010. PETSc users manual. Tech. Rep. ANL-95/11 - Revision 3.1, Argonne National Laboratory.

[7] Balay, S., Brown, J., Buschelman, K., Gropp, W. D., Kaushik, D., Knepley, M. G., McInnes, L. C., Smith, B. F., Zhang, H., 2011. PETSc web page. Http://www.mcs.anl.gov/petsc.

[8] Balay, S., Gropp, W. D., McInnes, L. C., Smith, B. F., 1997. Efficient management of parallelism in object oriented numerical software libraries. In: Arge, E., Bruaset, A. M., Langtangen, H. P. (Eds.), Modern Software Tools in Scientific Computing. Birkhäuser Press, pp. 163–202.

[9] Bernholdt D. (PI), 2010. TASCS Center. `http://www.scidac.gov/compsci/TASCS.html`.

[10] Buttler, H., AprilJune 2005. A guide to the python universe for esri users. ArcUser Mag.Available online at `http://www.esri.com/news/arcuser/`.

[11] CCA Forum, 2010. A hands-on guide to the Common Component Architecture. `http://www.cca-forum.org/tutorials/`.

[12] CSDMS, 2011. Community Surface Dynamics Modeling System (CSDMS). `http://csdms.colorado.edu`.

[13] CUAHSI, 2011. Consortium of Universities for the Advancement of the Hydrological Sciences Inc. . `http://www.cuahsi.org`.

[14] Dahlgren, T., Epperly, T., Kumfert, G., Leek, J., 2007. Babel User's Guide. CASC, Lawrence Livermore National Laboratory, UCRL-SM-230026, Livermore, CA.

[15] de St. Germain, J. D., Morris, A., Parker, S. G., Malony, A. D., Shende, S., May 15-17 2002. Integrating performance analysis in the Uintah software development cycle. In: Proceedings of the 4th International Symposium on High Performance Computing (ISHPC-IV). pp. 190–206. URL `http://www.sci.utah.edu/publications/dav00/ishpc2002.pdf`

[16] Diachin L. (PI), 2011. Center for Interoperable Technologies for Advanced Petascale Simulations (ITAPS). `http://www.scidac.gov/math/ITAPS.html`.

[17] EJB, 2011. Enterprise Java Beans Specification. `http://java.sun.com/products/ejb/docs.html`.

[18] ESMF Joint Specification Team, 2011. Earth System Modeling Framework (ESMF) Website. `http://www.earthsystemmodeling.org/`.

[19] FRAMES, 2011. Framework for Risk Analysis of Multi-Media Environmental Systems (FRAMES). `http://mepas.pnl.gov/FRAMESV1/`.

[20] Hill, C., DeLuca, C., Balaji, V., Suarez, M., da Silva, A., ESMF Joint Specification Team, 2004. The architecture of the Earth System Modeling Framework. Computing in Science and Engineering 6, 18–28.

[21] Hutton, E. W. H., Syvitski, J. P. M., 2008. Sedflux-2.0: An advanced process-response model that generates three-dimensional stratigraphy. Computers & Geosciences 34 (10), 1319–1337.

[22] Kessler, M. A., Anderson, R. S., Briner, J. P., 2008. Fjord insertion into continental margins driven by topographic steering of ice. Nature Geoscience 1, 365–369.

[23] Kettner, A. J., Syvitski, J. P. M., 2008. Hydrotrend version 3.0: a climate-driven hydrological transport model that simulates discharge and sediment load leaving a river system. Computers & Geosciences 34 (10), 1170–1183.

[24] Keyes D. (PI), 2011. Towards Optimal Petascale Simulations (TOPS) Center. http://tops-scidac.org/.

[25] Krishnan, S., Gannon, D., April 2004. XCAT3: A framework for CCA components as OGSA services. In: Proceedings of the 9th International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS 2004). IEEE Computer Society, pp. 90–97.

[26] Kumfert, G., April 2003. Understanding the CCA Specification Using Decaf. Lawrence Livermore National Laboratory.
URL http://www.llnl.gov/CASC/components/docs/decaf.pdf

[27] Larson, J. W., 2009. Ten organising principles for coupling in multi-physics and multiscale models. ANZIAM Journal 47, C1090–C1111.

[28] Larson, J. W., Norris, B., 2007. Component specification for parallel coupling infrastructure. In: Gervasi, O., Gavrilova, M. L. (Eds.), Proceedings of the International Conference on Computational Science and its Applications (ICCSA 2007). Vol. 4707 of Lecture Notes in Computer Science. Springer-Verlag, pp. 56–68.

[29] Lawrence Livermore National Laboratory, 2011. Babel. http://www.llnl.gov/CASC/components/babel.html.

[30] Lucas R. (PI), 2011. Performance Engineering Research Institute (PERI). http://www.peri-scidac.org.

[31] MathWorks, 2011. MATLAB - The Language of Technical Computing. http://www.mathworks.com/products/matlab/.

[32] Message Passing Interface Forum, 1998. MPI2: A message passing interface standard. High Performance Computing Applications 12, 1–299.

[33] NET, 2011. Microsoft .NET Framework. http://www.microsoft.com/net/.

[34] NetCDF, 2011. NetCDF. http://www.unidata.ucar.edu/packages/netcdf.

[35] OMP, 2011. Object Modeling System v3.0. http://www.javaforge.com/project/oms.

[36] Ong, E. T., Larson, J. W., Norris, B., Jacob, R. L., Tobis, M., Steder, M., 2008. A multilingual programming model for coupled systems. International Journal for Multiscale Computational Engineering 6, 39–51.

[37] Open Geospatial Consortium, 2011. OGC Standards and Specifications. http://www.opengeospatial.org/.

[38] Parker, G., 2011. 1d sediment transport morphodynamics with applications to rivers and turbidity currents. http://vtchl.uiuc.edu/people/parkerg/morphodynamic_e-book.htm.

[39] Peckham, S., 2008. Geomorphometry and spatial hydrologic modeling. Vol. 33 of Geomorphometry: Concepts, Software and Applications. Developments in Soil Science. Elsevier, Ch. 25, pp. 579–602.

[40] Peckham, S. D., Goodall, J. L., 2011. Driving plug-and-play components with data from web services: A demonstration of interoperability between CSDMS and CUAHSI-HIS. Computers & Geosciences (this issue).

[41] Rosen, L., 2004. Open Source Licensing: Software Freedom and Intellectual Property Law. Prentice Hall, `http://rosenlaw.com/oslbook.htm`.

[42] T. Oliphant et al., 2011. Scientific Computing Tools for Python – NumPy. `http://numpy.scipy.org/`.

[43] The MCT Development Team, 2006. Model Coupling Toolkit (MCT) Web Site. `http://www.mcs.anl.gov/mct/`.

[44] The OpenMI Association, 2011. The Open Modeling Interface (OpenMI). `http://www.openmi.org`.

[45] Tucker, G. E., Lancaster, S. T., Gasparini, N. M., Bras, R. L., 2001. The Channel-Hillslope Integrated Landscape Development (CHILD) Model. Academic/Plenum Publishers, pp. 349–388.

[46] United States Department of Energy, 2011. SciDAC Initiative homepage. `http://scidac.gov/`.

[47] VisIt, 2011. VisIt. `http://wci.llnl.gov/codes/visit`.

[48] XML, 2011. Extensible Markup Language (XML). `http://www.w3.org/XML/`.