Appendix 1: Sample Inventory of Modeling Courses

Maureen Berlin and Irina Overeem
July 2010

CSDMS Mission Statement:
The Community Surface Dynamics Modeling System (CSDMS) deals with the Earth's surface - the ever-changing, dynamic interface between lithosphere, hydrosphere, cryosphere, and atmosphere. We are a diverse community of experts promoting the modeling of earth surface processes by developing, supporting, and disseminating integrated software modules that predict the movement of fluids, and the flux (production, erosion, transport, and deposition) of sediment and solutes in landscapes and their sedimentary basins.

CSDMS Integration Facility staff is interested in making our products and tools accessible to help supplement existing college courses related to terrestrial, coastal, marine, hydrology, and carbonate topics. Based on a recommendation from the Education and Knowledge Transfer (EKT) Working Group meeting in Fall 2009, we conducted a survey of university course catalogs to learn how surface process modeling is currently being taught. An important caveat is that the results below are only as reliable as the course descriptions in university catalogs, which may be out of date, incomplete, or inaccurate. However, we posit that instructors who use modeling would most likely attempt to promote that feature of their course rather than obscure it, in order to help recruit students.

We targeted members of the Association of American Universities (AAU) (http://www.aau.edu/) to gain a representative sample of research-intensive universities. Rather than surveying all 63 AAU members, we chose a representative sample of 36 institutions from this list (Figure 1). We required that either a geology or civil engineering department was present for each surveyed university. Of the surveyed universities, 16 are also host institutions of CSDMS members (Figure 1).

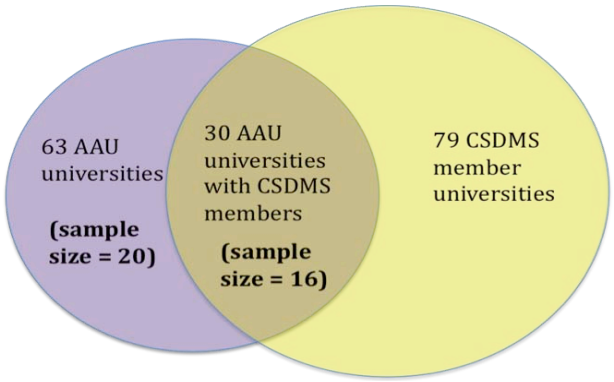


Figure 1. Overlap between AAU universities and CSDMS member universities.

All major regions of the U.S. are represented among the surveyed universities (including one Canadian university), although the Pacific Coast, Mid-Atlantic, and Midwest regions had higher concentrations, perhaps indicative of the larger population centers in those areas. Nineteen universities are public funded and seventeen are private institutions. The surveyed universities have undergraduate student populations that range in size from less than 1,000 to over 75,000, and the graduate student populations vary between just over 1,000 to more than 14,000 (Figure 2).

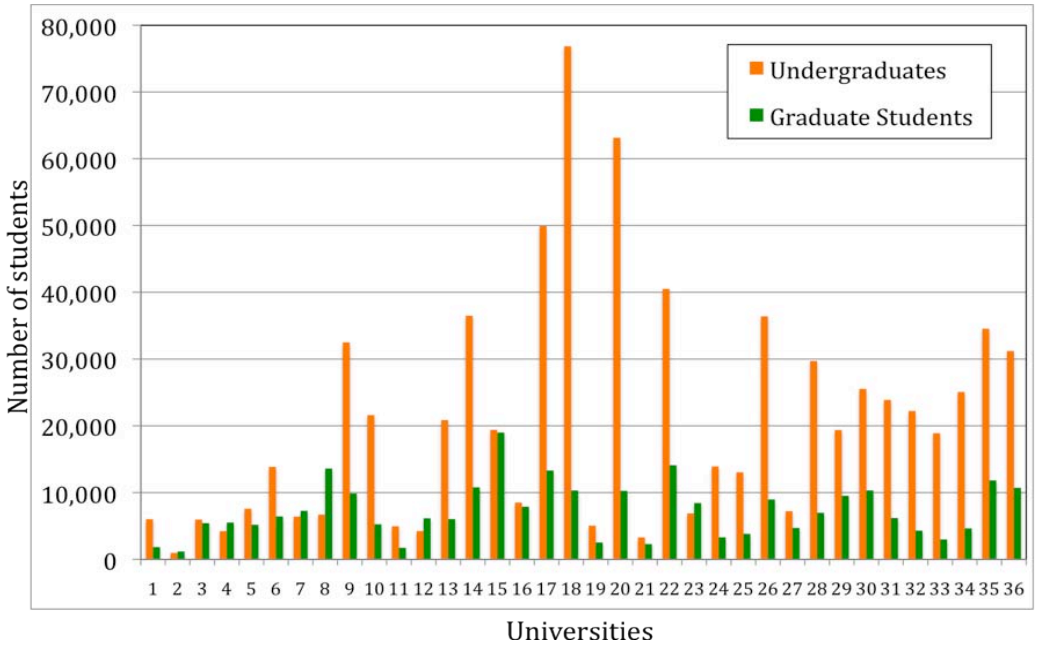


Figure 2. Range of size of student body populations at surveyed universities.

Box 1. Surveyed Universities

Brown University
California Institute of Technology
Carnegie Mellon University
Case Western Reserve University
Columbia University
Cornell University
Duke University
Harvard University
Indiana University (Bloomington)
Iowa State University
The Johns Hopkins University
Massachusetts Institute of Technology
McGill University
Michigan State University
New York University
Northwestern University
The Ohio State University
The Pennsylvania State University
Princeton University
Purdue University
Rice University
Rutgers, The State University of New Jersey
Stanford University
Stony Brook University-State University of New York
Syracuse University
Texas A&M University
Tulane University
The University of Arizona
University at Buffalo, The State University of New York
University of California, Berkeley
University of California, Davis
University of California, Irvine
University of California, Santa Barbara
University of Colorado at Boulder
University of Florida
University of Illinois at Urbana-Champaign

We reviewed course catalog descriptions and collected basic information such as course name, department, level, credit hours, and format. If a course was offered in multiple departments, we identified it either with the main host department, or the first instance of the course in the catalog. We noted any required prerequisites, programming languages used, and the objectives from the course description. We also tallied which courses might be especially relevant to any of the CSDMS working groups or focus research groups.

Course titles and keywords that were of interest include:

- modeling of earth surface processes
- GIS/remote sensing

- quantitative techniques/statistics
- sedimentary geology modeling
- hydrology/glaciology
- fluid dynamics
- groundwater hydrology/groundwater flow modeling/hydrogeology
- global change/climate modeling

We generally excluded those courses that were limited to the following topics, as these seemed peripheral to the types of modeling tools and educational products that CSDMS is developing:

- geophysics/geophysical or geological fluid dynamics/geodynamics
- paleoclimate
- pure computer science (e.g. C programming)
- general physical geology (too generic)
- numerical solutions of partial differential equations/numerical methods
- courses on high performance scientific computing or parallel computing
- statistics/time series analysis
- finite element modeling of geological materials/geotech/soil mechanics/soil science
- atmospheric modeling/meteorology (unless oceans are mentioned)
- geochemistry
- 1-credit seminars or reading seminars
- "Special Topics in…" with no course description (these may still be of interest to us, as evidenced by the upcoming CU course using CMT, which will be listed under Special Topics)
- field courses

For the 36 universities surveyed, we identified 1043 courses that had at least some relevance to CSDMS, either in terms of subject matter or in the use of modeling in the earth sciences. Of these courses, 717 were undergraduate level, 469 were graduate level, and 143 were cross-listed at both levels.

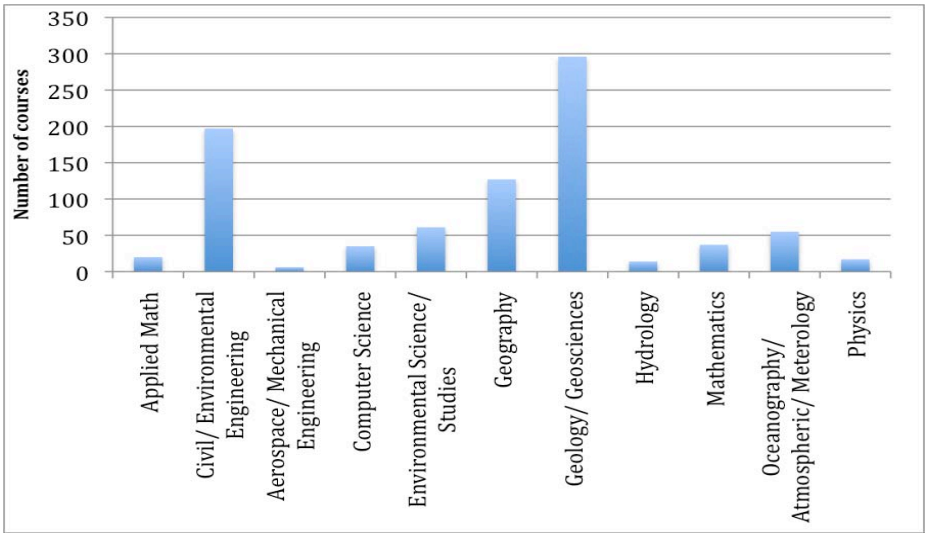


Figure 3. Frequency of all surveyed courses by university department.

Of the university departments offering CSDMS-relevant courses, Geology/Geosciences and Civil/Environmental Engineering Departments had the most courses (Figure 3). We also tallied which courses would have content that is relevant to CSDMS Working Groups and Focus Research Groups (and also mentioned climate, even though that is not a CSDMS group). Many of the surveyed courses can be connected to the Hydrology and Terrestrial Groups (Figure 4). It is interesting to note that although the

Terrestrial Working Group is the largest in terms of CSDMS membership and number of models, it does not correspond to the highest number of surveyed courses (Figure 4).

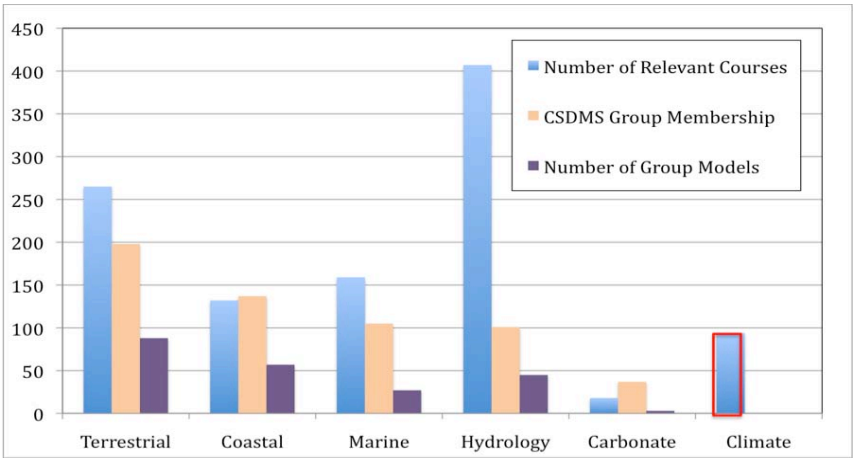


Figure 4. Comparison of frequency of courses relevant to CSDMS Focus Research Groups and Working Groups, membership within those groups, and number of group models as of July 2010.

Although CSDMS does currently focus on GIS or remote sensing services, we did tally the use of these software packages in addition to programming languages. Matlab was the most common language mentioned in course descriptions, although many Babel-supported languages (Fortran, C/C++, Python, and Java) were also indicated (Figure 5).

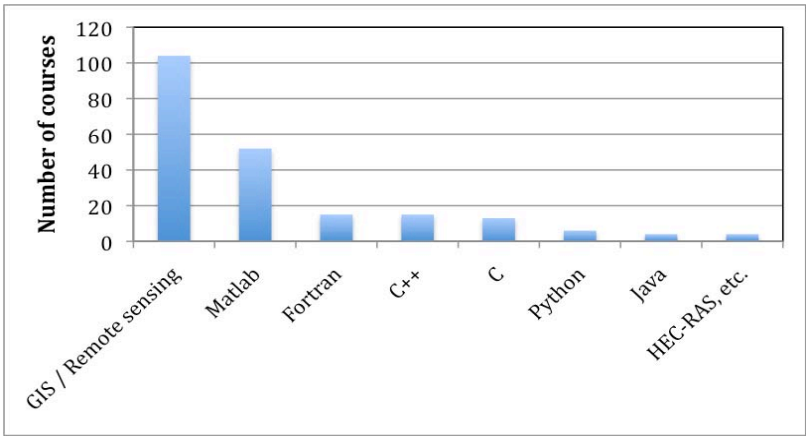


Figure 5. Frequency of courses that specified a software package or programming language.

In general, universities with larger undergraduate student body populations had more courses that we tracked as relevant to CSDMS (Figure 6). This trend did not hold with the graduate student population size (Figure 7).

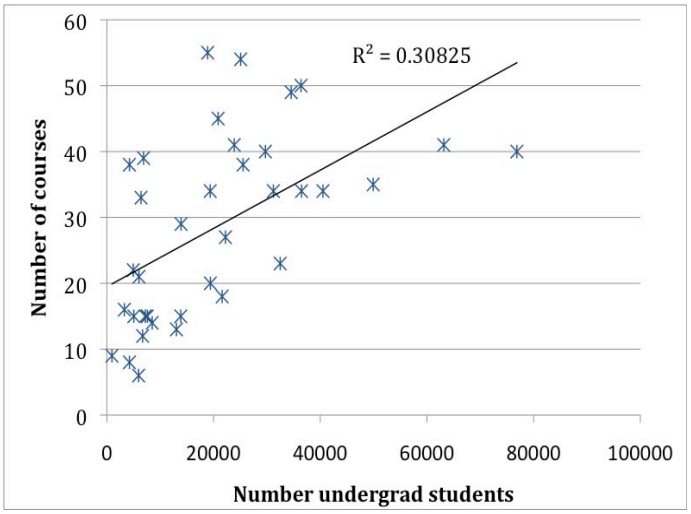


Figure 6. Number of surveyed courses vs. number of undergraduate students at each university.

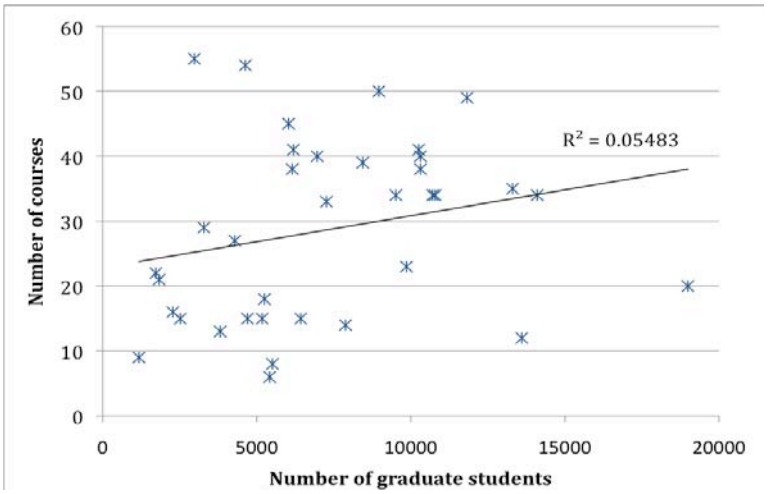


Figure 7. Number of surveyed courses vs. number of graduate students at each university.

On average, for each of the 36 surveyed universities, approximately 29 courses listed in the course catalogs would have some relevance to CSDMS modeling and educational efforts. The schools with the highest number of relevant courses were:

- University of California, Santa Barbara (55)
- University of Colorado, Boulder (54)
- Texas A&M (50)
- University of Florida (49)
- McGill (45)
- Purdue, University of California at Davis (41)
- Penn State, University of Arizona (40)

"High-Relevance" Courses:

During the process of gathering information from course catalogs, we made special note of courses that have "high relevance," based on the following criteria:

- "Modeling" is listed in course description
- Hands-on activities may be emphasized
- Quantitative problem solving using computers
- Courses that could directly use or benefit from CMT and other CSDMS Integration Facility products

We identified 167 courses that are highly relevant to CSDMS. Selected course titles for which the course descriptions met these criteria include:

- Physical Hydrology
- Coastal and Ocean Modeling
- Groundwater Modeling
- Computer Simulations in Earth and Planetary Sciences
- Geological Modeling
- Sequence Stratigraphy
- Sediment Transport and River Mechanics
- Advanced Watershed Hydrology
- Earth Systems Science
- Marine Sedimentology

The course descriptions for these high-relevance courses include some of the following elements:

- application of numerical analysis to mathematical modeling in the natural sciences
- watershed analysis, watershed hydrology with analytical and numerical models
- model building and validation; quantitative problems, forward and inverse modeling; model construction and simulation; computational analysis
- spatial/temporal modeling of water on landscapes
- scientific computing with science applications; software development for scientists
- environmental fluid mechanics and sediment transport with numerical models
- environmental quality numerical modeling
- computer simulation models of hydrology; hydrological forecast modeling
- quantitative modeling of fluids and sediments; sediment transport
- numerical modeling of coasts and oceans; model development for ocean circulation
- quantitative surface processes with numerical modeling
- simulations of hydrologic cycle; modeling hydrologic response to different climates
- mathematical modeling of river and coastal currents
- numerical modeling of groundwater flow; subsurface fluid flow; fate and transport of pollutants
- lake, river, coastal contaminant transport model development
- hydrologic/hydraulic computer modeling; surface water hydrology, floodplain hydraulics; 2D flow modeling; streamflow modeling
- developing numerical geoscience models
- use of computer programs for runoff calculation from catchments
- simulations of oceanic processes; advanced topics in modeling for ocean and estuarine environments using existing techniques and codes; numerical design of ocean models
- modeling of modern environmental problems
- computer-based methods of analysis in geomorphology; numerical models of sediment/debris flows
- hands-on applications using numerical modeling; numerical algorithms
- quantitative methods in natural resources and environmental sciences

Most universities have at least a few high-relevance courses, and on average, 15% of surveyed relevant courses at a given university were classified as highly relevant (Figure 8).

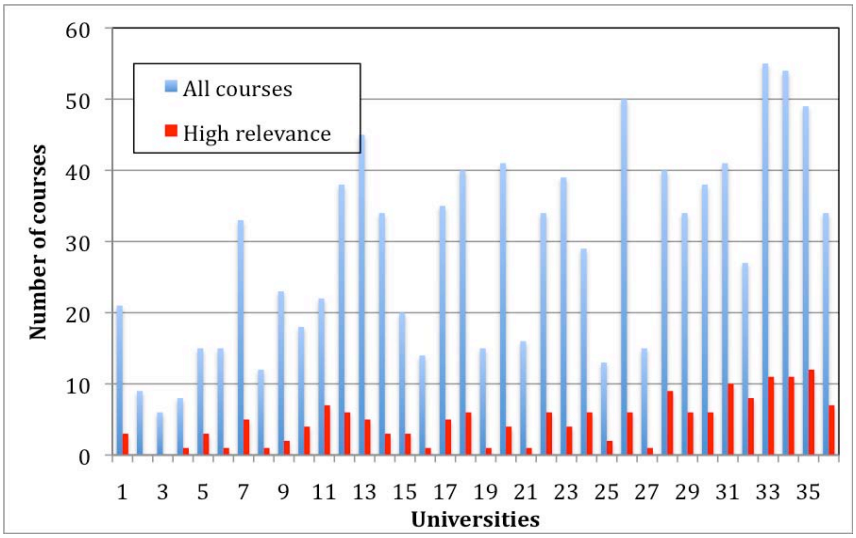


Figure 8. Frequency of all surveyed courses and high-relevance courses by university.

Civil and Environmental Engineering departments become the most popular, largely due to the presence of groundwater and surface water modeling courses (Figure 9). From this survey, we can speculate that hydrology courses represent an opportunity for the immediate or near-term use of CSDMS products, and that civil/environmental engineering departments may be the most logical host of these courses.

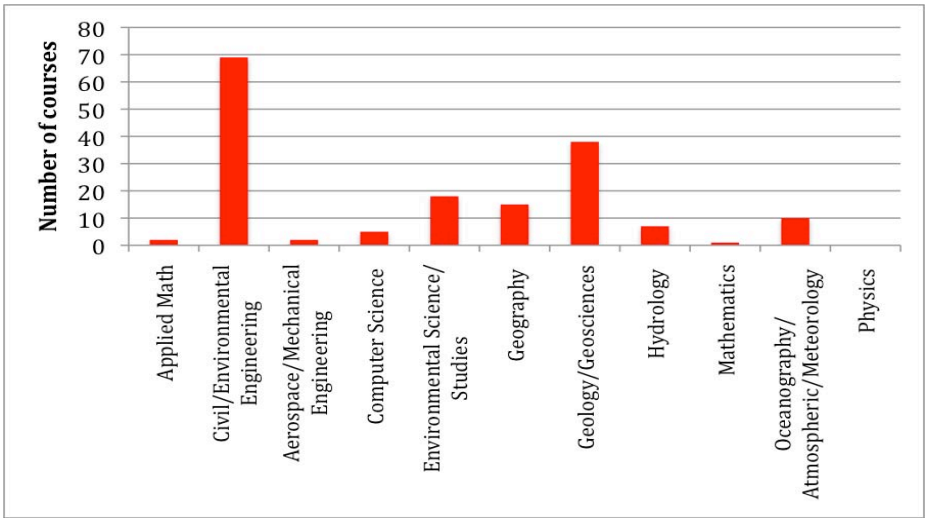


Figure 9. Frequency of high-relevance courses by university department.

The frequency of high-relevance courses was not well-correlated with size of either the undergraduate or graduate student populations (Figures 10, 11).

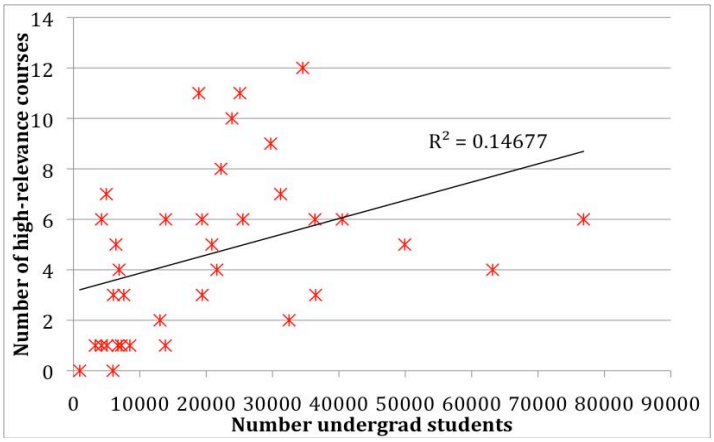


Figure 10. Number of high-relevance courses vs. number of undergraduate students for each surveyed university.

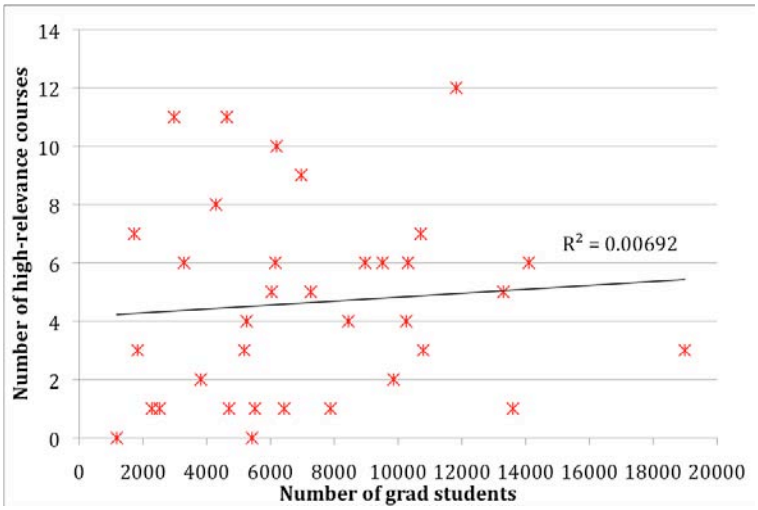


Figure 11. Number of high-relevance courses vs. number of graduate students for each surveyed university.

On average, for each of the 36 surveyed universities, approximately 4-5 courses listed in the course catalogs would be highly relevant to CSDMS modeling and educational efforts. The schools with the highest number of highly-relevant courses were:

- University of Florida (12)
- CU Boulder, UCSB (11)
- UC Davis (10)
- U. Arizona (9)
- UC Irvine (8)
- U Illinois, Johns Hopkins (7)
- SUNY Stony Brook, Rutgers, Penn State, UC Berkeley, SUNY Buffalo, Texas A&M, MIT (6)

To summarize, hydrology courses (groundwater, surface water) may represent the most immediate opportunity for use of CSDMS products. Modeling courses may be most common in civil/environmental engineering departments.

We intend to administer a course questionnaire to CSDMS members as a way to validate some of our results. We also need to consider how these results influence our modeling and educational products.

Appendix 2: CU Modeling Course Use Case

Maureen Berlin
July 2010

GEOL 5700: Surface Process Modeling: applying the CSDMS Modeling Tool
Instructors: Prof. James Syvitski, Dr. Irina Overeem, Dr. Scott Peckham.
2 credits, Fall 2010.

The CSDMS Modeling Tool (paired with the CSDMS wiki website) is used to support a semester-long two-credit course at the University of Colorado that centers on the use of numerical surface process models and hydrological models. Participants include three instructors and approximately ten *(?)* graduate students. Although the instructors and students will be in the same location during the class meeting times, remote access to the system and remote collaboration will be a central part of class participation. The course will involve both lectures and hands-on modeling.

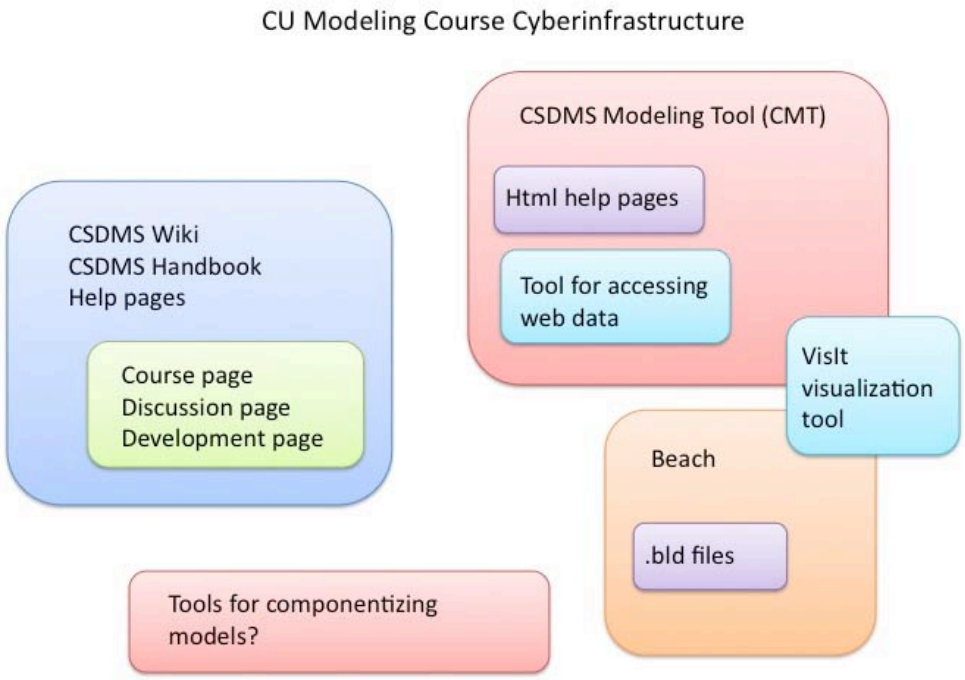
Course Description:
This course aims to familiarize earth sciences and engineering graduate students with a number of numerical surface process models and hydrological models available through CSDMS and set them up to use these tools for their own research purposes.

Goal:
At the end of the course, students should be able to design and run simulations for an independently designed research question within either the hydrological-glaciological, coupled river-delta, or stratigraphic domains.

Given the short timeline for course development (classes start Aug. 23), this use case was generally written to correspond with the existing CSDMS cyberinfrastructure (see diagram below):
-CMT and associated help files and VisIt
-directories on beach for sharing and storing files
-CSDMS wiki, including a course page and a discussion page
However, we should not restrict ourselves to this infrastructure in imagining and developing future use cases, and even here I've identified some capabilities that may or may not be currently in place.

Note that Instructor1 and Instructor2 are used interchangeably below. I've also avoided issues of homework or grading in favor of a more collaborative environment. A key theme to consider throughout is that students have the tools needed to adequately document their model runs, both to support student collaboration and instructor verification of completion of assignments.

The "How" following each paragraph attempts to identify the existing capability, or cyberinfrastructure, or summarize some of the software requirements that need to be put in place.



Phase 1. Preparation

Prior to the course, Instructor1 requests that all enrolled students join CSDMS and also obtain an account on beach. Obtaining access to beach through the University of Colorado will take 5-7 business days.
How: Students fill out forms on CSDMS wiki

Instructor2 creates a wiki page for the course and makes both Instructors administrators. Instructor2 posts basic course information, contact information for Instructors, a link to the course syllabus, and general references.
How: Instructors create a CSDMS wiki page with links to other documents. Syllabus should provide links to all associated wiki help pages, CSDMS Handbook sections, and tutorials for each section of the course.

Instructor1 posts links on the course page to relevant model questionnaires, CSDMS help pages, and other wiki pages for the course.
How: Model questionnaires must be available on the CSDMS wiki for each model that the course uses.

Instructor2 creates a discussion page on the CSDMS wiki, and posts an initial question for all participants: "Describe your previous modeling experiences and interests."
How: Create a discussion page on CSDMS wiki to save interactions and make them visible to others within the course; non-email discussion capability.
This capability is currently available as a "Talk" page—we just need to enforce the editing conventions. (http://www.mediawiki.org/wiki/Help:Talk_pages)
Does this page or ones that link from it need to be password-protected to allow conversations to be just seen by those inside the course? To what extent should the course results and progress be viewed by other CSDMS members and/or the general public? Need to be able to link to other documents and pdfs within the discussion.

Phase 2. Students Join the Course

Students join CSDMS (obtaining wiki access) and sign up for a beach account (obtaining CMT and beach access). While joining CSDMS the students can provide contact information, a link to their personal website, and other profile information.
How: Students fill out forms on CSDMS wiki

Instructor1 ensures that students have read permission to the course page, and read and write permission to the discussion page. This allows them to contribute posts to the discussion page while preserving the content of the main course page.
How: Instructors must be able to change permissions for students.

Students post responses to the instructor's question on the discussion page.
How: Students must be able to properly edit the discussion page.

Phase 3. Introduction to CSDMS and its High Performance Computing System.

Students launch CMT directly from the CSDMS web and login using their beach account. Students review html help files contained within CMT to become familiar with the CMT environment.
How: Students must have Java and VPN access.

Phase 4. Lectures on theory and applications of several models.

To support the lectures, Instructor1 provides links on the course page to model documentation and .bld-specific tutorials for several models currently available in CMT.
How: Instructors must be able to link from the course page to html help within CMT.

Instructor2 uploads pdfs of scanned journal articles or textbook chapters to a specific directory on beach (this is to limit copyright restrictions and avoid photocopying?)
How: Instructors and students must have access to a course directory on beach.

Phase 5. Lab exercises to explore 1) a coupled hydrological-glaciological model, i.e. TOPOFLOW and GC2D, 2) a coupled river-delta model, i.e. HydroTrend and CEM, 3) a stratigraphic model, SedFlux, and 4) a landscape evolution model, Erode.

Students load pre-determined configuration files (.bld) and practice doing model runs. Students click on the visualization tab and are able to create several plots of the data using an interactive GUI. They can compare these figures with example figures in the corresponding tutorials for each .bld file.
How: Students must have access to VisIt.

Students modify settings from the initial .bld file (such as swapping out components, or changing parameters within a component) and save a new configuration file for their particular model run. After running the new model, students save the .bld file and output files to a shared directory so other students may learn from these runs without having to recreate them. Students post the file path from their model runs, and associated model run metadata on the discussion page.
How: Students must be able to save output files into different directories.

Students create figures or movies from their model output files and upload these along with captions to potentially several places where the images can be viewed by other students, and/or reviewed and graded by the instructors.
-a personal page on the wiki
-the discussion page on the wiki
-a specific directory on beach
How: Students must have access to these components of the wiki, or be able to generate new html help pages.

Students use scripts to create new html files with their model figures and model run metadata (based on the model run configuration settings found in the associated .bld files). These pages can then be incorporated into CMT's set of help pages as metadata helpful for future CMT use.
How: Students must be able to generate new html help pages.

Students use the visualization tool to compare or overlay output files from their peers' model runs (using saved output files from one or more other students). They save several plots to the above listed places for discussion and grading purposes.
How: Students must have access to other students' model output. The discussion page should have a table or some way of logging when different model iterations are completed, so the class can keep track of this.

Students use components within CMT to ingest hydrologic or other data from a web server. They then use these data as input data for model runs.
How: Web-based data server access must be incorporated into CMT functionality, along with corresponding help documentation.

Phase 6. Design and run simulations for an independently designed research question within one of these modeling domains.

Working within the four domains above (e.g., using existing CMT components), students either build model configuration files from scratch or modify pre-existing .bld files to explore different research questions. Students will likely need to generate new input files. Students generate figures and movies as above and post to several places.
How: Students must be able to create and import input files for their model runs; documentation and tools must be present.

Instructor1 fields complaints that students are having difficulty with their model runs, or notices that the results students are posting have errors. They are able to examine how the model was run, make changes to the configuration, and rerun the simulation.
How: Instructors must be able to access .bld files and log files from student model runs, and then modify them.

Phase 7. Course Wrap-Up

In addition to university-led Faculty Course Questionnaires, Instructors administer an optional, anonymous survey to students regarding their experience in the course (e.g., joys/frustrations while using CMT, goals or barriers to future involvement with CSDMS, suggestions for improvements to future courses).
How: Anonymous survey form on the wiki?

Instructor1 archives student results either in a cleaned-up discussion page, or as new individual html help files that can be used by future users of CMT.
How: Instructors should be able to copy selected student .bld files over to the main example directories for CMT.

Appendix 3: Recommended Protocols for Model Software Developers

James P.M. Syvitski, Community Surface Dynamics Modeling System (CSDMS) Integration Facility,
Eric Grunsky, Natural Resources Canada, Geological Survey of Canada, 601 Booth St., Ottawa, K1A 0E8, Canada and Editor-in Chief of Computers & Geosciences.

Abstract

Developers of research grade Geoscience models should ensure that their software contributions follow these protocols: 1) Hold an open-source 'GPL v2' or a 'GPL v2 compatible' license; 2) Be widely available to the community of scientists through an international model or code repository (e.g. CSDMS or C&G); 3) Undergo a level of peer review; 4) Be written in an open-source language, or have a pathway for use in an open-source environment; 5) Where appropriate, be written or refactored to allow for componentization by having an interface, with exchange items documented; 6) Be accompanied with a formal metadata file, along with test files; 7) Be clean and well-documented. Software may be vetted at three levels: 1) the model behaves as advertised; 2) the code meets pre-approved specifications or follows community protocols; and 3) the model provides for an acceptable depiction of nature. Freely available and open-source code allows for complete information transfer and replication of results — the foundation of modern science. Open source allows for the original developer to be recognized, protected, and their software to have the greatest impact on science.

Introduction

At the 2009 International Association of Mathematical Geosciences (IAMG) annual meeting at Stanford University, representatives of Community Surface Dynamics Modeling System (CSDMS) and IAMG met to review protocols adopted by CSDMS as a possible guide for code submission to IAMG's journal Computers & Geosciences (C&G). Here we review this discussion and argue for protocol adoption beyond CSDMS and C&G, and for the wider Geoscience community. The paper details concepts related to code sharing in general using community modeling concepts as a guide.

About the Community Surface Dynamics Modeling System

CSDMS is an integrated community of experts who promote the quantitative modeling of earth-surface processes. CSDMS develops, supports, and disseminates integrated software modules that involve the Earth surface — the dynamic interface between lithosphere, atmosphere, cryosphere, and hydrosphere. CSDMS coordinates a growing community of more than 78 U.S. Academic Institutions, 17 US Federal labs and agencies, 67 non-U.S. institutes from 20 countries, and companies within an industrial consortium. CSDMS serves this diverse community by promoting the sharing and re-use of high-quality, open-source modeling software. The CSDMS Model Repository in January 2010 comprised a searchable inventory of more than 170 models with more than 3 million lines of code. CSDMS also offers the Geoscience community a Model-coupling Framework, a Data Repository related to the CSDMS mission, and a CSDMS Education portal.

About the journal Computers & Geosciences

C&G features research articles and application articles that describe new computation methods for the geosciences: e.g. computational infrastructure, informatics, collection, representation, management, analysis, visualization, as well as for software development and scientific and social use of Geoscience information and review articles and short notes are also accepted to support this general mission.

The aims of CSDMS and C&G overlap a great deal, and while both have complementary missions, they serve the community in different ways, the latter concentrating on peer-reviewed journal papers that may or may not be accompanied by open-source software. Code submitted to C&G is presently archived and made available to the community through an IAMG portal. In the past, code submissions were not always reviewed as part of the normal review of the submitted paper. In that sense code submitted to the C&G repository (http://www.iamg.org/CGEditor/index.htm) is simply parked at their portal ready for downloading. Although program code is typically tested by reviewers, no rigorous procedures are in place with specific criteria for formal testing. Questions related to downloaded code remain a private affair between a reader and the author(s).

CSDMS protocols for model contribution

Protocols are the procedures or the system of rules governing contributed community software, and provide both technical and social recommendations to model developers. Software contributions to the CSDMS Model Repository should:

1) Hold an open-source license.

2) Be widely available to the community of scientists.

3) Receive a level of vetting, for example the software should be determined to do what it says it does.

4) Be written in an open-source language, or have a pathway for use in an open-source environment.

5) Be written or refactored to allow for componentization by having an interface, with specific I/O exchange items documented.

6) Be accompanied with a formally defined metadata file, along with test files.

7) Be clean and documented using keywords within comment blocks to provide basic metadata for the model and its variables.

These protocols provide extensibility to software and allow for state-of-the-art tools to convert stand-alone models into flexible, "plug-and-play" components that can be assembled into larger applications (Syvitski et al., in press). The protocols also allow a migration pathway towards high-performance computing (HPC). We describe each protocol below.

Open-Source Software license

The usage and redistribution of software is defined by its software license. Software licenses come in a range of variety including proprietary, free and/or open source. Code may be distributed as an executable or as source code. Proprietary licenses control the usage or redistribution, and the copyright remains with the publisher. Proprietary software often involves commerce, made available in closed-source binary format, with legally binding use- or view-restrictions.

A free, open source license in contrast allows the software code to be: 1) inspected, 2) modified, and 3) redistributed. The GNU General Public License (GPL) also allows the original or modified version of the software to be commercially sold, even if the code remains freely available. Open source licensing requires that the source code be available. The GPL v2 license is widely used by free open source software developers and because the license:

- Provides a better quid-pro-quo for developers
- Establishes collaboration between people
- Protects the developers work
- Encourages increasing the amount of free software.

Using the GNU GPL license requires that all the released improved versions be free software. This means you can avoid the risk of having to compete with a proprietary modified version of your own work. A developers' project is likely to be more successful if it accommodates fellow developers who also use the GPL license. CSDMS urges program developers to choose 'GPL v2' or a 'GPL v2 compatible' license to make it possible to couple the model with other models such that other people can use them. Below we list approved licenses by the Free Software Foundation (FSF) that are GPL v2 compatible:

- Artistic License 2.0
- Berkeley Database License
- modified BSD license
- Boost Software License
- Cryptix General License
- Eiffel Forum License version2

- GNU Lesser General Public License
- Intel Open Source License
- ISC license
- MIT license
- Python Software Foundation License 2.0.1, 2.1.1 and newer
- W3C Software Notice and License
- zlib/libpng license
- Zope Public License version 2.0

To maximize software use by fellow scientists and to make it free software, the following lines of notice must be incorporated into the program, attached to the start of each source file to most effectively convey the exclusion of warranty. Each file should have the "copyright" line and a pointer to where the full notice is found:

1. <one line to give the program's name and a brief idea of what it does.>
2. Copyright (C) <year> <name of author>
3. Developer can be contacted by <email> and <paper mail>
4. This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.
5. This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.
6. A copy of the GNU General Public License is available through the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Further details of how to license software, and to use employer signatures, can be found at http://csdms.colorado.edu/wiki/License. CSDMS Integration software is licensed under the BSD or MIT-X11 license. This implies that permission is granted, free of charge, to any person obtaining a copy of CSDMS integration software and associated documentation files, without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the software, and to permit persons to whom the software is furnished, subject to the following conditions:

> THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

Many scientists are strong believers that science is advanced through mutual cooperation. Community modeling involves the collective efforts of individuals that code, debug, test, document, run, and apply models and modeling frameworks. Community modeling relies on open-source code to address the practical need of contributing developers to examine and modify the code. Open-source code provides complete information transfer. This *transparency* is important because code is the ultimate statement of the scientific hypotheses embodied in a numerical model, and their implementation. In the world of software, *details are important*. A scientific article describing code, as is often the case with Computers & Geosciences articles, may provide the theoretical equations, but the solution to these equations can take numerous forms, and each solution has its pyramid of assumptions and limitations. Therefore open-source code allows for full *peer review* and *replication of results* — the foundation of modern science.

If a geologist was to map outcrops in a certain part of our landscape, and subsequently write a science article about their findings, another geologist sooner or later could go to the same landscape and determine whether the original data and interpretations were legitimate and appropriate. Peer review is as important in the science of software engineering as it is in the observational sciences.

Open source code allows for reuse, often in new and clever ways. This certainly reduces redundancy. In the U.S., Congressional law dictates that software developed with public funds must become publicly available, with national security exceptions. Open-source code is an effective way to meet this requirement. CSDMS promotes the development of free open source code since it operates largely with public funds in the public domain. Yet even industry supports the CSDMS open-source efforts.

Open source does not mean that the original developer is not recognized. Developers are recognized with the metadata associated with each model, with GPL2 software license protection, through community exposure, vetting and recognition, and through accelerated citations within peer-reviewed publications.

Software Availability

In the world of science, software code is often considered "research grade". That means that it is often relatively untested, may contain bugs, and might not be at the standards required for true "commercial grade" or "operational grade" code. Coding glitches in research grade code are often unknown by the original author. With wider community use, such problems are discovered and either rectified by the original author or the bug discoverer. Commercial grade code is widely available, limited to the details related to the financial transaction and other proprietary redistribution and use restrictions. Operational grade code describes code used by governments for monitoring or enforcing, and may or may not be widely available to the public. The Weather and Research Forecasting (WRF) model is widely adopted by weather services worldwide to make operational predictions. WRF code is open source. Other operational models, such as NOAA's WAVEWATCH III®, an ocean wave model used for hindcasting, nowcasting and forecasting, is subject to U.S. export restrictions. The code is open source and widely available, but a short list of countries are not allowed access.

Research grade code should be widely available to the community of scientists. The best way to have the code available is through an appropriate international repository. Geoscience models can be submitted to the CSDMS Model Repository. Code associated with articles in Computers & Geosciences can be submitted to its C&G Code Repository. Since the code is open source, it can appear in more than one Repository. Too often code is issued with an open source license (or no license at all) but access to the code is restricted to access through the author. Unfortunately this allows the author to decide who they will give the code to. This runs contrary to the transparency needed in science, and we discourage this level of availability.
Some models such as ROMS, the Regional Ocean Modeling System, support a very large community (1000s) who use and develop the model. New users must register through the ROMS portal. We view this level of access as acceptable as the ROMS developers need to demonstrate a large user group, to those who fund their program activity. ROMS is open source and free without restrictions to all legitimate scientists.

Vetting Software

This is perhaps the most difficult subject of all of the CSDMS protocols. Vetting is the review and approval process of, in our case, Geosciences software. Vetting comes in many forms, from informal to formal, and from objective to subjective. Software vetting has three components: 1) verification that the model behaves as advertised; 2) confirmation that the code meets pre-approved specifications, for example is accompanied with metadata documentation or meets community protocols; and 3) demonstration that the model provides an accurate depiction of nature. Vetting in the context of C&G would mean that the software was subjected to some level of independent peer-review although there are no formal rules in place for reviewers.

In the world of community modeling, it is not unusual for software to be reviewed by a working group of specialists. Reviewers would be given a set of guidelines and standard questions, and would be asked to the test the model and respond to the queries. The review officer behaves like a journal editor in the sense that identified problems might require fixes before being given the stamp of approval of the community. For a journal such as C&G, the process might involve a reviewer to provide a similar analysis. The reviewer could be independent of the paper review, or might agree to do both activities on behalf of the community and journal, even though this extends beyond the current review mandate of the journal.

Answering whether a model offers an accurate depiction of nature is complex — science is provisional, and a yes/no answer is often not possible. This is true with all scientific manuscripts to some degree, code being no different. After determining that the model does what it says it does, the reviewer might reflect on the level of testing that lies behind the model. For example and when appropriate, has the model been run against known benchmark experiments, and compared with field or laboratory observational data? Are the claims associated with the model within acceptable uncertainties related to the time and space resolutions of the model (or other appropriate resolutions)?

More subjective are questions of performance that often relate to how the conservation equations are solved. Performance often translates as the usefulness to an end user. In the field of fluid dynamics, performance varies with each level of complexity: advection-diffusion, shallow water wave equation, Reynolds-averaged Navier Stokes, large-eddy simulation, direct numerical simulation, hydrostatic, non-hydrostatic, Boussinesq, non-Boussinesq. Ultimately transparency trumps subjectivity.

Open-Source Programming Language

Computer software is written in a programming language that is able to access a target compiler to allow precise translation between source code and object code — an 'executable' able to run on a particular computer platform. In general all source code is written in a higher-level computer language and the executable is written in machine code. Programming languages are static only in narrow release formats. There are many versions of Fortran, for example, with new versions having greatly enhanced abilities to work with modern platforms and compilers and their libraries for enhanced functionality.

In the open source community, developers develop their models using an open-source language (e.g. C, C++, any Fortran, Java, Python), or a language that has a pathway for use in an open-source environment. A developer should test whether their code can compile using an open source compiler (e.g. GNU Fortran compiler). This will ensure the greatest chance of portability of the code from one computational platform to another, minimizing problems.

The CSDMS community requires its code to be written in an open-source language so that the various models can communicate with each other using 'Babel'. Babel is an open-source, language interoperability tool (and compiler) that automatically generates the "glue code" that allows components written in different computer languages to communicate (Dahlgren et al. 2007). Babel currently supports C, C++, Fortran 77, 90, 95 & 2003, Java and Python. Almost all of the Geosciences models held in the CSDMS Model Repository are written in one of these languages. Babel enables the passing of variables with data types that may not normally be supported by the target language (e.g. objects, complex numbers). To create the glue code needed components written in different programming languages to pass information between them, Babel only needs to know about the interfaces of the components. It does not need any implementation details. Babel can ingest a description of an interface in one of two "language neutral" forms, XML (eXtensible Markup Language), or SIDL (Scientific Interface Definition Language). SIDL provides a description of a scientific software component interface, including the names and data types of all arguments and the return values for each member function.
Software written in other high-level languages might have a translation pathway to one of the BABEL-supported open-source programming languages. For example, CSDMS offers the community an enhanced version of 'i2py' designed to convert IDL source code to the open-source Python language.

Refactoring a Model into a Component

Most models are written to be stand-alone models. In other words, the software is designed to define and initialize its variables and arrays, read in any needed input data, run the program to get realizations according to its discretized algorithms, write out its output, and end the run. In the field of environmental science, a model would cover a given domain, for example lake dynamics. After some time, the model may be further developed to cover other environmental domains, so for example a lake model might gain a river basin model. Large codes often involve more than one environmental process or domain, for example wind-driven currents plus wave dynamics in oceanography, or channelized flow overland flow and groundwater flow in

hydrology. Codes that involve multiple domains often involve a diversity of experts needed for their development, and thus the birth of community modeling. Inevitably when the codes reached a certain level of complexity, the codes became modeling frameworks. Too large for individuals to understand all the details, developers would pass on their process modules to be implemented by a master(s) of the code.

Modern software engineering has developed new standards for data exchange, model interfaces, and ways to employ varied computational platforms (laptops, servers, high performance computing clusters, distributed or cloud computing). In the world of community Geosciences modeling, there has been strong movement towards developing models as components within architectures and frameworks, each offering interfaces, exchange items (Syvitski et al., in press). Below we introduce these terms and show how the CSDMS community has adopted these concepts. While these concepts may not be appropriate for all contributions to the journal Computers and Geosciences, they are highly appropriate for developers wishing to enter the world of community modeling.

Frameworks increase a developer's productivity, and a user's functionality. Environmental modeling frameworks support the coupling of models into functional units (e.g. components, classes, or modules), component interaction and communication, time stepping, regriding of arrays, scaling of spatial data, multiprocessor support, and cross language interoperability. A framework may also provide a uniform method of trapping or handling exceptions (i.e. errors).

An Architecture is the set of standards that allow components to be combined and integrated for enhanced functionality, for instance on high-performance computing systems. The standards are necessary for the interoperation of components developed in the context of different frameworks. Software components that adhere to these standards can be ported with relative ease to another compliant framework.

Components are functional units that once implemented in a particular framework are reusable in other models within the same framework, with little migration effort. One advantage of using a modeling framework is that pre-existing components can be reused to facilitate model development. Component-based modeling brings about the advantages of "plug and play" technology. Component programming builds upon the fundamental concepts of object-oriented programming, with the main difference being the presence of a framework. Components are generally implemented as classes in an object-oriented language, and are essentially "black boxes" that encapsulate some useful bit of functionality. A framework provides the environment wherein components can be linked together to form applications. A component differs from an ordinary subroutine, module or class, because they can communicate with other components written in a different programming language.

Components typically provide one or more interfaces by which a caller can access their functionality. In the context of plug-and-play components, the word interface refers to a named set of member functions (methods), defined with regard to argument types and return types but without any actual implementation. An interface is a user-defined type, similar to an abstract class, with member function "templates" but no data members. A component contains an actual implementation for each member function (and possibly member functions beyond the ones that comprise a particular interface). Therefore it is possible and often useful for a single component to expose multiple, different interfaces, allowing a component to be used in a greater variety of settings.

Most surface dynamics models advance values forward in time on a grid or mesh and have a similar internal structure. This structure consists of lines of code before the beginning of a time loop (the initialize step), lines of code inside the time loop (the run step) and finish with additional lines after the end of the time loop (the finalize step). Virtually all component-based modeling efforts (e.g. ESMF, OpenMI, OMS, CSDMS) recognize the utility of moving these lines of code into three separate functions, with names such as Initialize, Run and Finalize, or **IRF** for short (Syvitski et al., in press). These three IRF functions constitute a simple model-component interface that provides a calling program with fine-grained access to a model's capabilities and the ability to control its overall time stepping so that it can be used in a larger application. The calling program "steers" a set of components and so is referred to as a driver.

A meaningful linkage of components often requires both data exchange and IRF functions. A model's interface must also describe functions that access data that it wishes to provide (getter functions) and methods that allow other components to change its data (setter functions). With getter and setter interface functions, connected components can query generated data as well as alter data from the other model. Component connections are made through 'provides ports' and 'uses ports' within a Common Component Architecture framework (Armstrong et al. 1999). The first provides an interface to the component's own functionality (and data). The second specifies a set of capabilities (or data) that the component requires from another component to complete its task. A provides-port that exposes an IRF interface, allows another component to gain access to its initialize, run, and finalize steps. The uses-port presents functionality that it lacks itself and therefore requires from another component. The component is not able to function until it is connected to a component that has the required functionality. This allows a model developer to create a new model that uses the functionality of another component without having to know the details of that component or to even have that component exist at all.

This style of plug-and-play component programming benefits both model programmers and users. Within a framework model developers are able to create models within their areas own of expertise and rely on experts outside their field to fill in the gaps. Models that provide the same functionality can easily be compared to one another simply by unplugging one model and plugging in another, similar model. In this way users can easily conduct model comparisons and more simply build larger models from a series of components to solve new problems.

For example standalone models are made into component models by dividing them into tasks that other component models could use (Fig. 1).

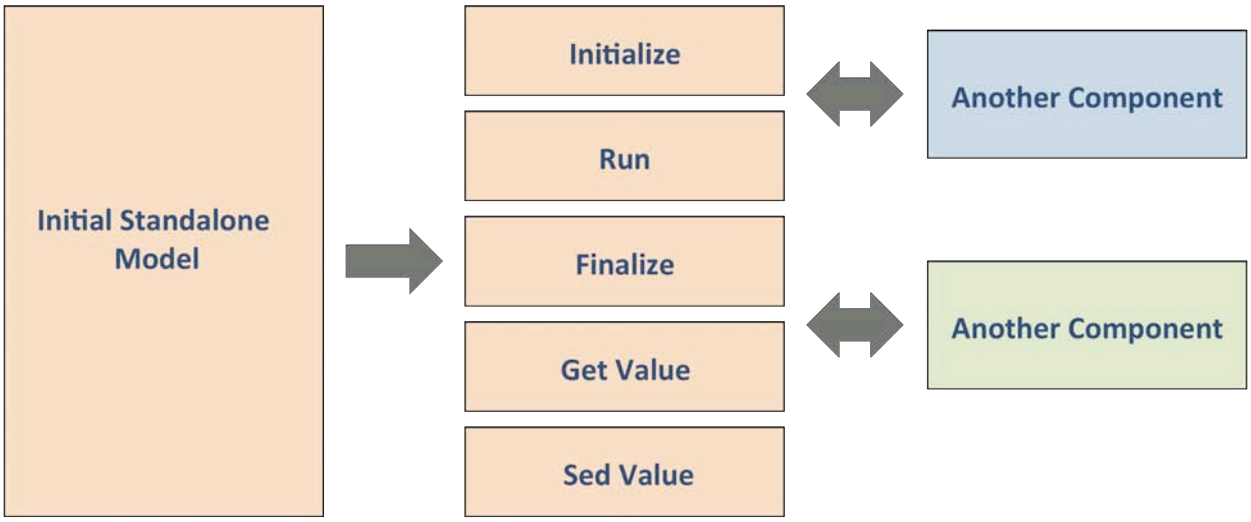


Figure 1. Refactoring a stand-alone model for linkage to other model components.

Once a contributed model has been refactored into a component model, it becomes available to be linked to other appropriate models within the CSDMS component library to provide value added products beyond the intention or domain of the original model (Fig. 2). The language neutral compiler BABEL allows for models to communicate across various languages (Fig. 2). Access to CCA/CSDMS, OpenMI and ESMF Services, such as grid remapping tools, is then made available. Databases and files can also be componentized and coupled within the CSDMS framework.

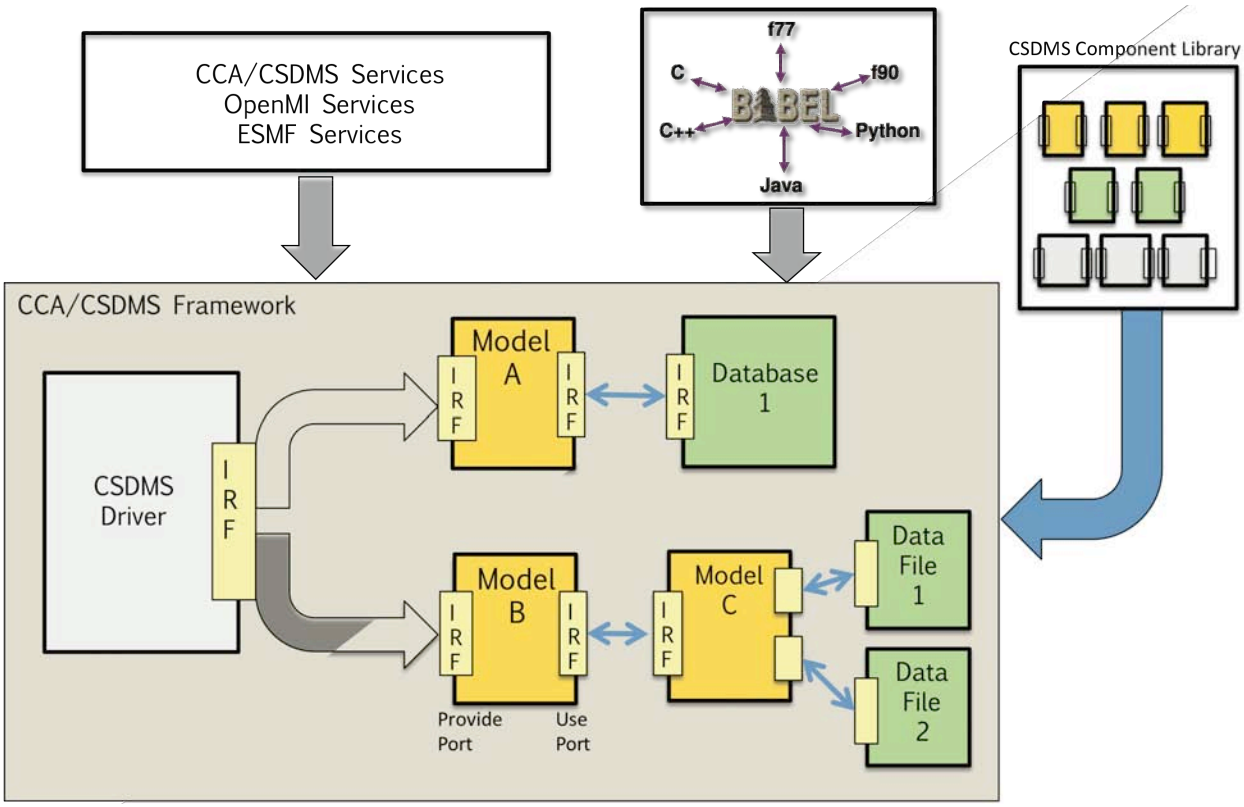


Figure 2. The CSDMS model coupling domain.

Metadata Description File

Information that describes contributed software is a necessary requirement for any code submission to either CSDMS or C&G. Appropriate metadata should cover contact information on the code developer(s), and information on the model: model domain, spatial dimensions (e.g. 2Dxz), and spatial extent (e.g. regional scale). The model description, if it is not already described in a paper, should include processes represented, key equations and key parameters, length scale and resolution constraints, time scale and resolution constraints, and numerical limitations and issues. Technical description should include: supported platforms, programming language, code optimization (e.g. parallel computing), development period, code availability and repository, software license, framework or interface compliance, memory requirements, and typical run times. Metadata should also include a description of the input and output files, including their format and whether pre- or post-processing is needed, and type of visualization software that is required. Unless described in an accompanying paper, the level of testing should be described. Input files to run the model and output files to verify the initial model run should also be included with the metadata.

Clean and Documented Code

Submissions to a model or code Repository should be refactored for maintenance and extensibility (Fowler, 1999; Kerievsky, 2004). Extraneous source lines that have been commented out should be removed. Code should be well documented both for future developer readability and to eliminate future mistakes. Where possible source code should be annotated using keywords within comment blocks to provide basic metadata for the model and its variables. Units should be well defined.

Summary

While the CSDMS protocols are mission oriented, they also offer good practice for code submission to the Computers & Geosciences Repository, and for code development in general. CSDMS protocols have been widely vetted within its extensive community (Hutton et al., 2010), and among other affiliated modeling

communities (Voinov et al., 2010). The protocols provide developers with recognition and protection, increased longevity and usability of the source code, and greater penetration into the community of a model development or its accomplishments. The protocols reflect the increased level of accountability required by funders. They eliminate duplication and further the advance and enhance science. We recommended these protocols for code submission to Computers and Geosciences

Acknowledgments

The discussion on vetting includes comments from Professor P Wiberg (U Virginia), Professor B Murray (Duke U), Professor G Tucker (U Colorado Boulder), and Professor R Slingerland (Penn State U). The discussion on components and frameworks includes comments from Dr. O David (Colorado State U), Dr. S Peckham (CSDMS Boulder), and Dr. C. Delucca (CIRES/NOAA Boulder). The discussion on licenses and refactoring includes comments from Dr. A Kettner and Dr. E Hutton (CSDMS Boulder).

References

Armstrong, R., D. Gannon, A. Geist, et al. 1999. Toward a common component architecture for high-performance scientific computing. In *Proceedings of the 8th Intl. Symposium on High Performance Distributed Computing*, pp. 115-124.

Collins, N., G. Theurich, C. DeLuca, et al. 2005. Design and implementation of components in the Earth System Modeling Framework. *Intl. J. High Performance Computing Applications* 19: 341-350.

Dahlgren, T., T. Epperly, G. Kumfert and J. Leek 2007. *Babel User's Guide. 2007 edition.* Center for Applied Scientific Computing, U.S. Dept. of Energy and University of California Lawrence Livermore National Laboratory, 269 pp.

Fowler, M. 1999. *Refactoring: Improving the design of existing code*. Addison Wesley Professional, Reading MA, 439 pp.

Gregersen, J.B., Gijsbers, P.J.A., and Westen, S.J.P. 2007. OpenMI : Open Modeling Interface. *J. Hydroinformatics* 9: 175-191.

Hutton, E.W.H., J.P.M. Syvitski & S.D. Peckham, 2010. Producing CSDMS-compliant Morphodynamic Code to Share with the RCEM Community. In *River, Coastal and Estuarine Morphodynamics RCEM 2009*, eds. C. Vionnet et al. Taylor & Francis Group, London, ISBN 978-0-415-55426-CRC Press, p. 959-962.

Kerievsky, J. 2004. Refactoring to Patterns. Addison Wesley Professional, Reading MA, 367 pp.

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

Voinov, C. DeLuca, R. Hood, S. Peckham, C. Sherwood, J.P.M. Syvitski, 2010, A community approach to Earth systems modeling. *EOS Transactions of the AGU*, 91(13): 117-124.