The Technology Behind the Community Surface Dynamics Modeling System (CSDMS)

Scott D. Peckham
Chief Software Architect for CSDMS

October 10, 2008

csdms.colorado.edu

CSDMS

Community Surface Dynamics Modeling System

CSDMS Education and Knowledge Transfer (EKT) Working Group Meeting, Boulder, CO

NSF

Why is it Difficult to Link Models?

1. Written in different languages (conversion is time-consuming, error-prone and snapshots do not keep up with developer's updates).
2. 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.
3. Models may have different dimensionality (1D, 2D or 3D)
4. Models may use different grid types (rectangles, triangles, polygons)
5. Each model has its own time loop or "clock".
6. The numerical scheme may be either explicit or implicit.
7. The type of coupling required poses its own challenges. Some common types of model coupling are: (a) ***Layered*** = A vertical stack of grids (e.g. distributed hydrologic model), (b) ***Nested*** = Usually a high-res model embedded within (and driven by) a lower-res model. (e.g. regional winds/waves driving coastal currents, or a 3D channel flow model within a landscape model), (c) ***Boundary-coupled*** = Model coupling across a natural (possibly moving) boundary, such as a coastline.

CSDMS
Community Surface Dynamics Modeling System

Functional Specs for the CSDMS

Support for **multiple operating systems**
(especially Linux, Mac OS X and Windows)

Support for **parallel computation** (multi-proc., via MPI standard)

Language interoperability to support code contributions written in C & Fortran as well as more modern object-oriented languages (e.g. Java, C++, Python) (CCA is language neutral)

Support for both **legacy code** (non-protocol) and more structured code submissions (“procedural” and object-oriented)

Should be **interoperable with other coupling frameworks**

Support for both structured and unstructured grids

Platform-independent GUIs and graphics where useful

Large collection of **open-source tools**

CSDMS
Community Surface Dynamics Modeling System

Scientific “Coupling Frameworks”

ESMF (Earth System Modeling Framework)

www.esmf.ucar.edu, maplcode.org/maplwiki

PRISM (Program for Integrated Earth System Modeling)

www.prism.enes.org (uses OASIS4)

OpenMI (Open Modeling Interface)

www.openmi.org (an interface standard vs. framework)

CCA (Common Component Architecture)

www.cca-forum.org,

www.llnl.gov/CASC/components/babel.html

Others: GoldSim (**www.goldsim.com**) commercial

FMS (**www.gfdl.noaa.gov/~fms**) GFDL

Non-scientific ones include CORBA, .NET, COM, JavaBeans, Enterprise Java Beans (see Appendix slide for links)

Overview of CCA

Widely used at DOE labs (e.g. LLNL, ANL, Sandia) for a wide variety of projects (e.g. fusion, combustion)

Language neutral; Components can be written in C, C++, Fortran 77/90/95/03, Java, or Python; supported via a compiler called **Babel**, using SIDL / XML metadata

Interoperable with ESMF, PRISM, MCT, etc.

Has a rapid application development tool called **BOCCA**

Similar to CORBA & COM, but science application support

Can be used for single or multiple-processor systems, distributed or parallel, MPI, high-performance (HPC)

Structured, unstructured & adaptive grids

Has stable DOE / SciDAC (www.scidac.gov) funding

Key CCA Concepts & Terms

Architecture = A software component technology ***standard*** (e.g. CORBA, CCA, COM, JavaBeans. synonym: "component model")

Framework = Environment that holds CCA components as they are connected to form applications and then executed. Provides a small set of standard ***services***, available to all components. May also provide a ***language interoperability tool*** (e.g. Babel). The framework can be tailored to the type of high-performance computing, e.g. ***Ccaffeine*** for parallel and ***XCAT*** for distributed. Others are SCIRun2 and Decaf.

Components = Units of software functionality (black boxes) that can be connected together to form applications ***within a framework***. Components expose well-defined interfaces to other components.

Interface = As defined in Java, similar to an abstract class. A specific collection of class ***member functions*** or ***methods***, with data types specified for all arguments and return values but ***no implementation***.

Ports = CCA's term for component interfaces, either ***uses*** or ***provides***.

Example: Basic “IRF” Interface

A component is often implemented as a class with a set of member functions or methods that provide a caller with complete control over the component’s capabilities. One benefit of this is that the caller can use its own time loop or clock instead of the one the model uses in stand-alone mode. This makes it easier to combine the capabilities of multiple models in a larger model.

Initialize() = Open & read input files, initialize variables, open output files.

Run_Step() or Execute() = Run a single “step”, which may be a time step or an iteration step (e.g. root-finding step or relaxation step).

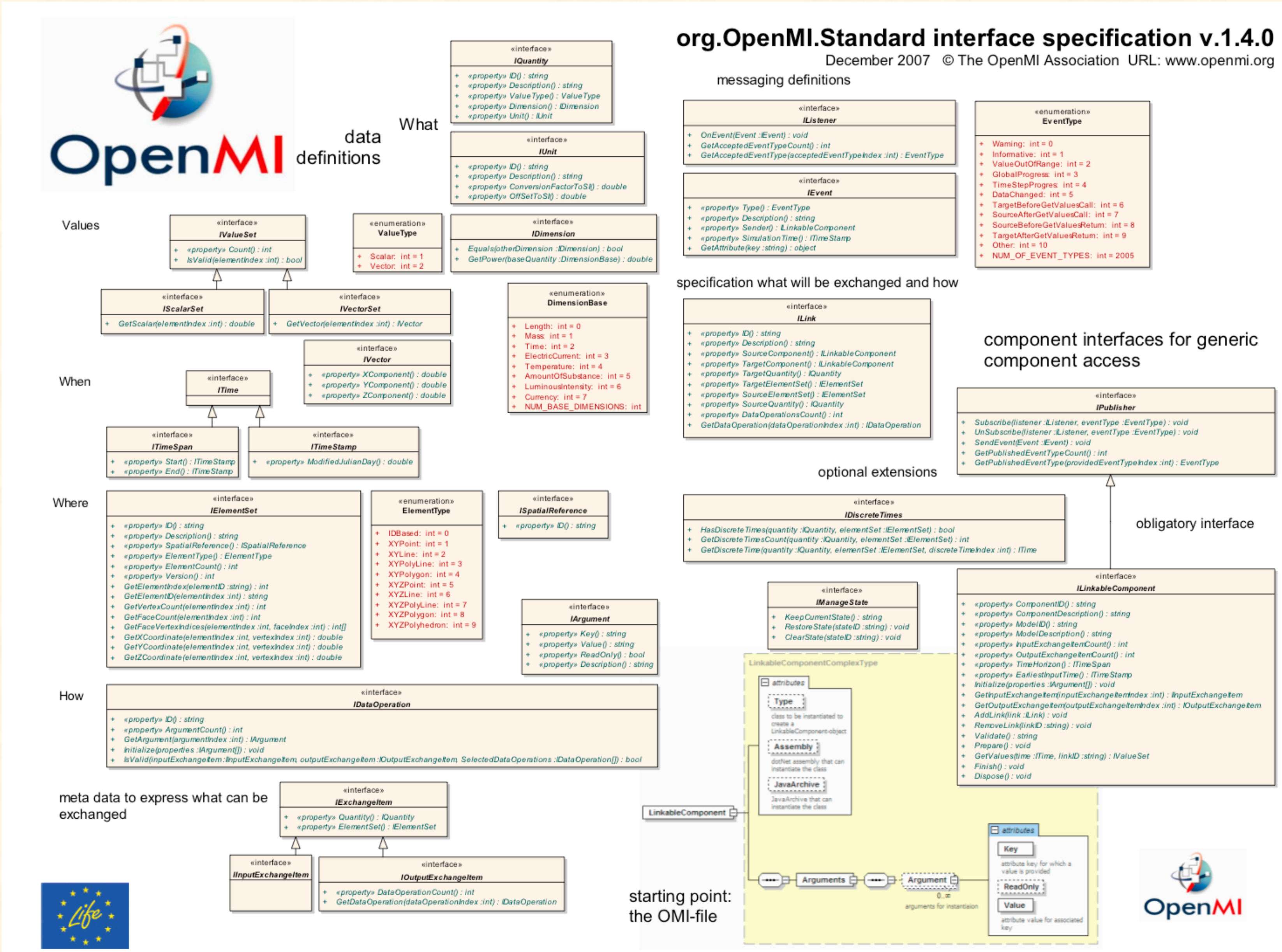
Get_Values() = What, When, Where and How. Return a specified variable at a specified time. Can also specify which grid cells and data operation.

Finalize() or Cleanup() = Close all files, print messages, free memory.

Test() = Perform one or more tests, from sanity check to comparison with an analytic solution.

Run_Model() = Run the entire model in “stand-alone” mode, using information from an input file.

Example: OpenMI Interface



Some Key CCA Tools

Babel = A "multi-language" compiler for building HPC applications from components written in different languages. (**http://www.llnl.gov/CASC/components/babel.html**)

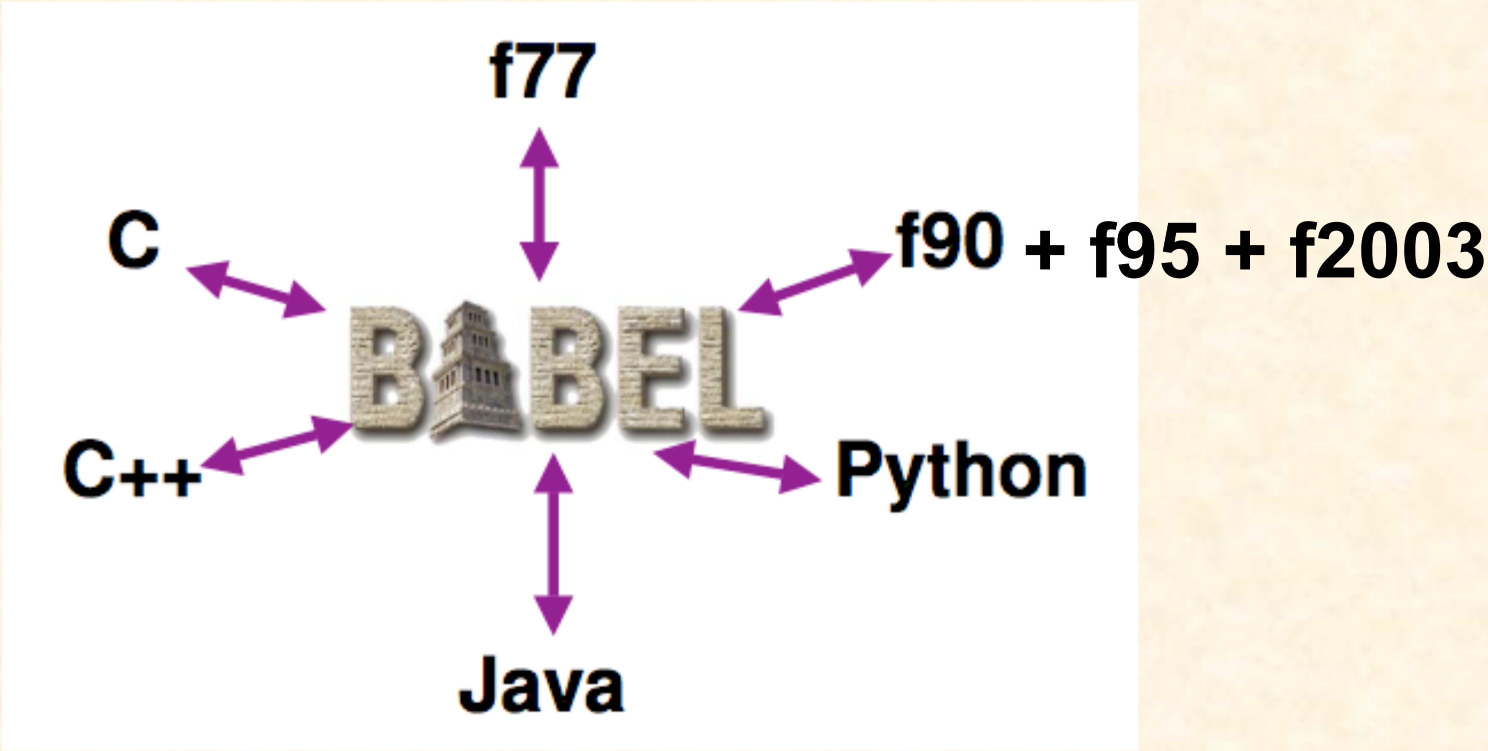
SIDL = Scientific Interface Definition Language (used by Babel). Allows language-independent descriptions of interfaces.

Bocca = A user-friendly tool for rapidly building applications from CCA components (RAD = Rapid Application Development) (**http://portal.acm.org/citation.cfm?id=1297390**)

Ccaffeine = A CCA component framework for parallel computing (**http://www.cca-forum.org/ccafe/ccaffeine-man**)

New CCA build system = Unnamed, user-friendly build system for the complete CCA "tool chain". It uses a Python-based tool called Contractor.

CCA: The Babel Tool



Language interoperability is a powerful feature of the CCA framework. Components written in different languages can be rapidly linked in HPC applications with hardly any performance cost. This allows us to "shop" for open-source solutions (e.g. libraries), gives us access to both procedural and object-oriented strategies (legacy and modern code), and allows us to add graphics & GUIs at will.

CCA: The Babel Tool

Minimal performance cost: A widely used rule of thumb is that environments that impose a performance penalty in excess of 10% will be summarily rejected by HPC software developers.

Babel's architecture is general enough to support **new languages**, such as Matlab, IDL and C# once bindings are written for them.

More than a least-common-denominator solution; it **provides object -oriented capabilities** in languages like C, F77, F9X where they aren't natively available.

Has intrinsic support for **complex numbers** and flexible **multi-dimensional arrays** (& provides for languages that don't have these). Babel arrays can be in row-major, column-major or arbitrary ordering. This allows data in large arrays to be transferred between languages without making copies.

Babel opens scientific and engineering libraries to a wider audience.

Babel **supports RPC** (remote procedure calls or RMI) over a network.

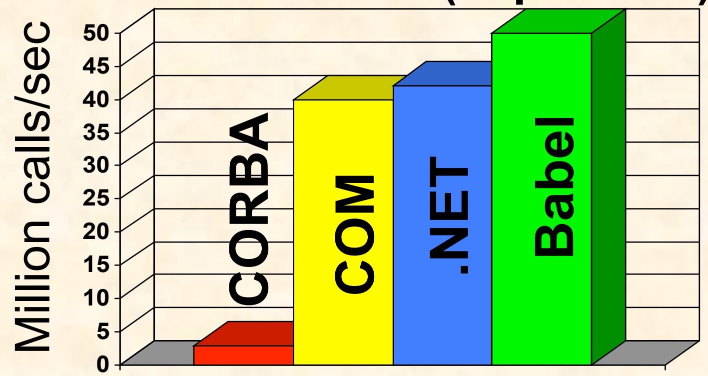
CCA: The Babel Tool

BABEL is Middleware for HPC

2006

"The world's most rapid communication among many programming languages in a single application."

Performance (in process)



	CORBA	COM	.NET	Babel
BlueGene, Cray, Linux, AIX, & OS X	No	No	No	Yes*
Fortran	No	Limited	Limited	Yes
Multi-Dim Arrays	No	No	No	Yes
Complex Numbers	No	No	No	Yes
Licensing	Vendor Specific	Closed Source	Closed Source	Open Source

CCA: The Bocca Tool

Provides project management and comprehensive build environment for creating and managing applications composed of CCA components

The purpose of Bocca is to let the user create and maintain useful HPC components without the need to learn the intricacies of CCA (and Babel) and waste time and effort in low-level software development and maintenance tasks. Can be abandoned at any time without issues.

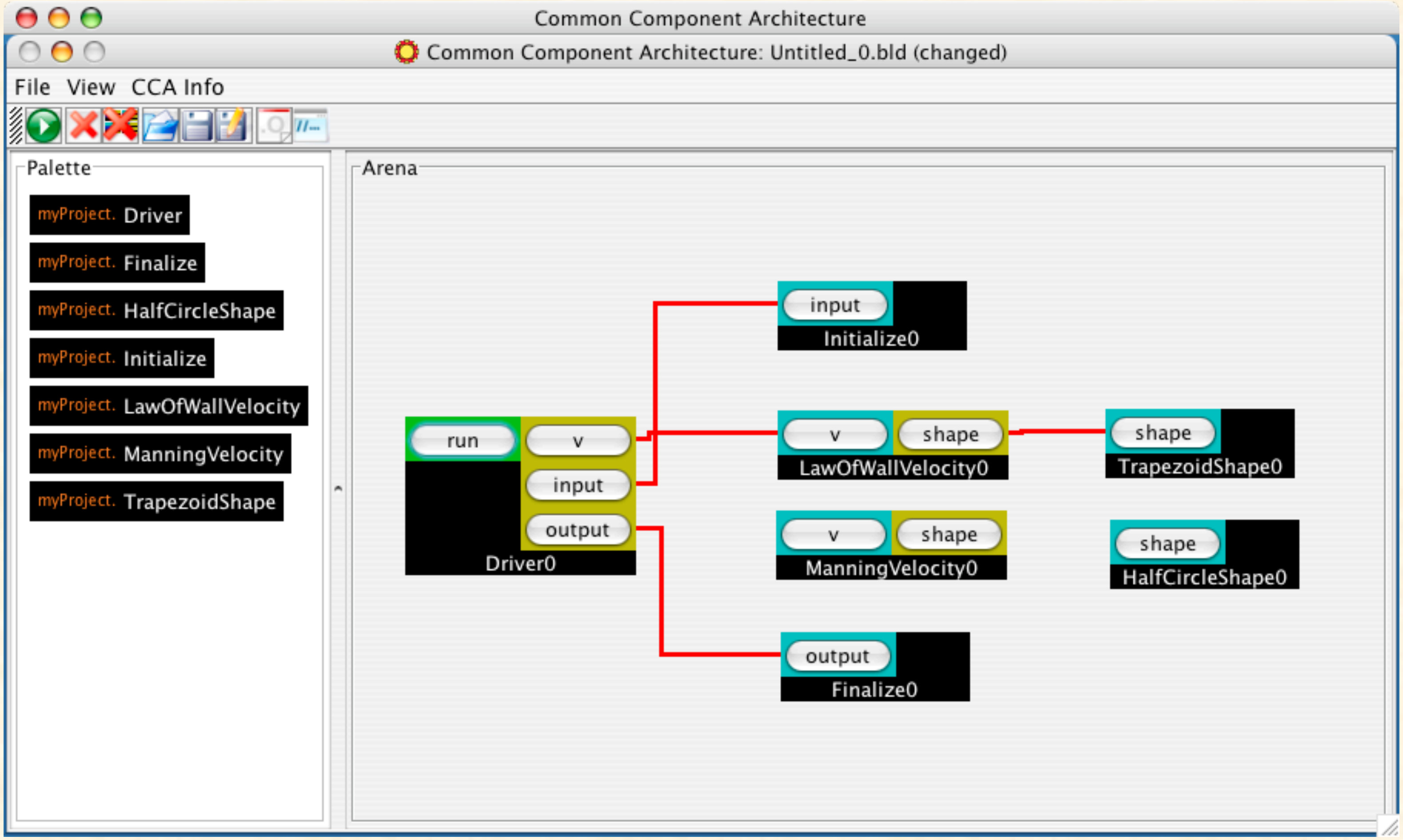
Bocca lays down the scaffolding for a complete componentized application without any atttendant scientific or mathematical implementation.

Built on top of Babel; is language-neutral and further automates tasks related to component “glue code”

Supports short time to first solution in an HPC environment

Easy-to-make, stand-alone executables coming in March 2008
(automatically bundles all required libraries; RC + XML -> EXE)

CCA: The Ccaffeine-GUI Tool



A "wiring diagram" for a simple CCA project. The CCA framework called **Ccaffeine** provides a "visual programming" GUI for linking components to create working applications.

Requirements for Code Contributors

1. Code must be in a Babel-supported language.
2. Code must compile with a CSDMS-supported, open-source compiler (e.g. gcc, gfortran, etc.)
3. Refactor source code to have an ***IRF interface***
4. Provide descriptions of all input & output ***exchange items***
5. Include suitable ***testing procedures*** and data
6. Include a user's guide or at least basic documentation
7. Specify what ***open-source license*** applies to your code
8. Use standard or generic file formats whenever possible for I/O
9. Apply a CSDMS automated wrapping tool

CSDMS
Community Surface Dynamics Modeling System

Other CCA-Related Projects

CASC = Center for Applied Scientific Computing
(https://computation.llnl.gov/casc/)

TASCS = The Center for Technology for Advanced Scientific Computing Software
(http://www.tascs-scidac.org) (focus is on CCA and associated tools; was CCTTSS)

PETSc = Portable, Extensible Toolkit for Scientific Computation
(http://www.mcs.anl.gov/petsc) (focus is on linear & nonlinear PDE solvers; HPC/MPI)

ITAPS = The Interoperable Technologies for Advanced Petascale Simulations Center
(http://www.itaps-scidac.org) (focus is on meshing & discretization; was TSTT)

PERI = Performance Engineering Research Institute
(http://www.peri-scidac.org) (focus is on HPC quality of service & performance)

TOPS = Terascale Optimal PDE Solvers
(http://www.scidac.gov/ASCR/ASCR_TOPS.html) (focus is on solvers)

SCIRun = CCA framework from Scientific Computing and Imaging Institute
(http://software.sci.utah.edu/scirun.html) (this is a CCA framework)

Conclusions

The Common Component Architecture (CCA) is a mature and powerful environment for component-based software engineering (CBSE) and building high-performance computing (HPC) applications.

Some of its most powerful tools include Babel, Bocca, Ccafe-GUI and the Ccaffeine framework. Each of these tools fulfills a particular need in an elegant manner in order to greatly simplify the effort that is required to build an HPC application.

The CCA framework currently meets most of the requirements of CSDMS and native Windows support (vs. Cygwin) is likely in the near future.

CCA has been shown to be interoperable with ESMF and should also be interoperable with a Java version of OpenMI.

For more information, please see the "CSDMS Handbook" at:
http://csdms.colorado.edu/wiki/index.php/Tools_CSDMS_Handbook

Python Support in CCA / Babel

Support for **Java & Python** makes it possible to add components with GUIs, graphics or network access anywhere in the application (e.g. via **wxPython or PyQT**). Python code can be compiled to Java with **Jython**. (See **www.jython.org** for details)

NumPy is a fairly new Python package that provides fast, array-based processing similar to Matlab or IDL. **SciPy** is a closely related package for scientific computing. **Matplotlib** is a package that allows Python users to make plots using Matlab syntax.

Python is used by Google and is the new ESRI scripting language. It can be expected that this will result in new GIS-related packages/ plug-ins. Python is entirely open-source and a large number of components are available (e.g. XML parser). Currently has over one million users and is growing.

GIS tools are often useful for earth-surface modeling and visualization.

Component Technology

Advantages of Component vs. Subroutine Programming

Can be written in different languages and still communicate.

Can be replaced, added to or deleted from an app. at run-time via dynamic linking.

Can easily be moved to a remote location (different address space) without recompiling other parts of the application (via RMI/RPC support).

Can have multiple different interfaces and can have state.

Can be customized with configuration parameters when application is built.

Provide a clear specification of inputs needed from other components in the system.

Have potential to encapsulate parallelism better.

Allows for multicasting calls that do not need return values (i.e. sending data to multiple components simultaneously).

CBSE = Component-Based Software Engineering

Component technology is basically "plug and play" technology (think of "plugins")

With components, clean separation of functionality is mandatory vs. optional.

Facilitates code re-use and rapid comparison of different methods, etc.

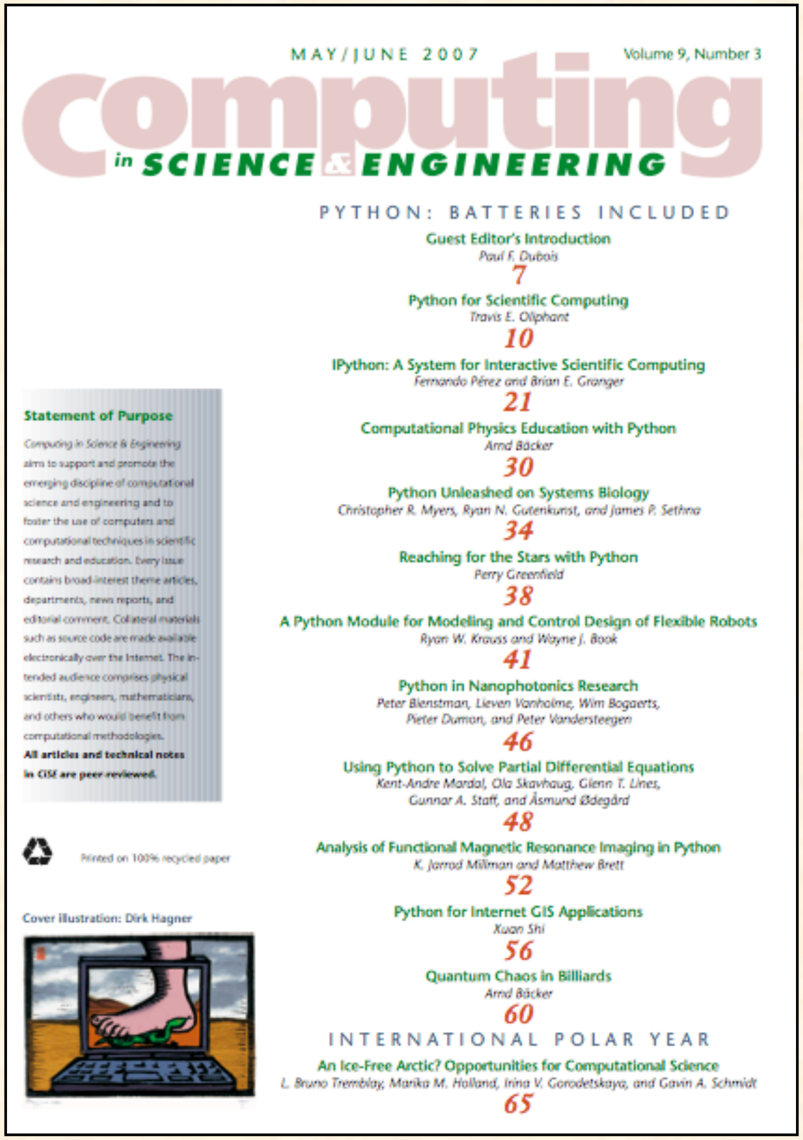
Facilitates efficient cooperation between groups, each doing what they do best.

Promotes economy of scale through development of community standards.

Possible Component Examples

Airy Waves	Stokes Waves	SWAN	REF-DIF	Boussinesq (FunWave)
Bagnold	Einstein	Meyer-Peter Muller	Yalin	Power law
Green-Ampt	Smith-Parlange	Beven	Richards' 1D	Richards' 3D
Kinematic Wave	Diffusive Wave	Dynamic Wave		
MARSSIM (Howard)	CHILD (Tucker)	SIBERIA (Willgoose)	Erode (Peckham)	TOPOG (CSIRO)
GEOTOP (Rigon)	GSSHA (Ogden)	MMS (Leavesley)	TopoFlow (Peckham)	MIKE SHE (DHI)

All approaches to modeling a given "process" or phenomenon are wrapped to present a standard "plug-and-play", object-oriented interface for their common capabilites (as method functions of some class)



MAY/JUNE 2007 Volume 9, Number 3

computing in SCIENCE & ENGINEERING

PYTHON: BATTERIES INCLUDED

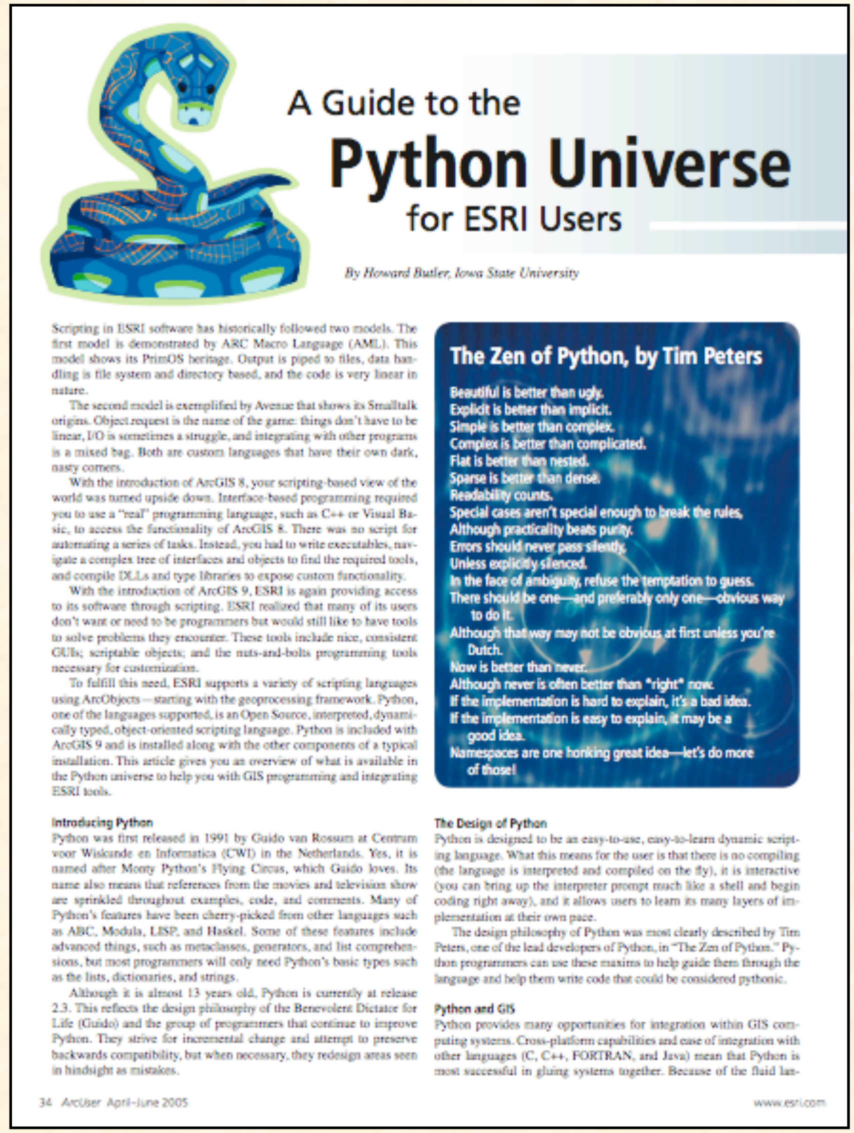
Statement of Purpose

Computing in Science & Engineering aims to support and promote the emerging discipline of computational science and engineering and to foster the use of computers and computational techniques in scientific research and education. Every issue contains broad-interest theme articles, departments, news reports, and editorial comment. Collateral materials such as source code are made available electronically over the Internet. The intended audience comprises physical scientists, engineers, mathematicians, and others who would benefit from computational methodologies. **All articles and technical notes in CiSE are peer-reviewed.**

Printed on 100% recycled paper

Cover illustration: Dirk Hagner

Python: Batteries Included, special issue of "Computing in Science & Engineering devoted to Python, May-June 2007, vol. 9(3), 66 pp. Nice collection of articles, incl. papers on ipython, matplotlib, GIS, solving PDEs.



A Guide to the Python Universe for ESRI Users

By Howard Butler, Iowa State University

Scripting in ESRI software has historically followed two models. The first model is demonstrated by ARC Macro Language (AML). This model shows its PrimOS heritage. Output is piped to files, data handling is file system and directory based, and the code is very linear in nature.

The second model is exemplified by Avenue that shows its Smalltalk origins. Object.request is the name of the game: things don't have to be linear, I/O is sometimes a struggle, and integrating with other programs is a mixed bag. Both are custom languages that have their own dark, nasty corners.

With the introduction of ArcGIS 8, your scripting-based view of the world was turned upside down. Interface-based programming required you to use a "real" programming language, such as C++ or Visual Basic, to access the functionality of ArcGIS 8. There was no script for automating a series of tasks. Instead, you had to write executables, navigate a complex tree of interfaces and objects to find the required tools, and compile DLLs and type libraries to expose custom functionality.

With the introduction of ArcGIS 9, ESRI is again providing access to its software through scripting. ESRI realized that many of its users don't want or need to be programmers but would still like to have tools to solve problems they encounter. These tools include nice, consistent GUIs; scriptable objects; and the nuts-and-bolts programming tools necessary for customization.

To fulfill this need, ESRI supports a variety of scripting languages using ArcObjects—starting with the geoprocessing framework. Python, one of the languages supported, is an Open Source, interpreted, dynamically typed, object-oriented scripting language. Python is included with ArcGIS 9 and is installed along with the other components of a typical installation. This article gives you an overview of what is available in the Python universe to help you with GIS programming and integrating ESRI tools.

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one—and preferably only one—obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea—let's do more of those!

Introducing Python

Python was first released in 1991 by Guido van Rossum at Centrum voor Wiskunde en Informatica (CWI) in the Netherlands. Yes, it is named after Monty Python's Flying Circus, which Guido loves. Its name also means that references from the movies and television show are sprinkled throughout examples, code, and comments. Many of Python's features have been cherry-picked from other languages such as ABC, Modula, LISP, and Haskel. Some of these features include advanced things, such as metaclasses, generators, and list comprehensions, but most programmers will only need Python's basic types such as the lists, dictionaries, and strings.

Although it is almost 13 years old, Python is currently at release 2.3. This reflects the design philosophy of the Benevolent Dictator for Life (Guido) and the group of programmers that continue to improve Python. They strive for incremental change and attempt to preserve backwards compatibility, but when necessary, they redesign areas seen in hindsight as mistakes.

The Design of Python

Python is designed to be an easy-to-use, easy-to-learn dynamic scripting language. What this means for the user is that there is no compiling (the language is interpreted and compiled on the fly), it is interactive (you can bring up the interpreter prompt much like a shell and begin coding right away), and it allows users to learn its many layers of implementation at their own pace.

The design philosophy of Python was most clearly described by Tim Peters, one of the lead developers of Python, in "The Zen of Python." Python programmers can use these maxims to help guide them through the language and help them write code that could be considered pythonic.

Python and GIS

Python provides many opportunities for integration within GIS computing systems. Cross-platform capabilities and ease of integration with other languages (C, C++, FORTRAN, and Java) mean that Python is most successful in gluing systems together. Because of the fluid lan-

34 ArcUser April–June 2005 www.esri.com

Butler, H. (2005) A guide to the Python universe for ESRI users, ArcUser (April-June 2005), p. 34-37. (tools for ellipsoids, datums, file formats like shapefiles)

Other Component Architecture Links

(Commercial, non-HPC, non-scientific computing)

CORBA (Object Management Group)
http://www.omg.org/gettingstarted
http://www.omg.org/gettingstarted/history_of_corba.htm

COM (Component Object Model, Microsoft, incl. COM+, DCOM & ActiveX)
http://www.microsoft.com/com/default.mspx

.NET (Microsoft Corp.)
http://www.microsoft.com/net

JavaBeans (Sun Microsystems)
http://java.sun.com/products/javabeans

Enterprise JavaBeans (Sun Microsystems)
http://java.sun.com/products/ejb

Types of Model Coupling

Layered = A vertical stack of grids that may represent:

(1) different domains (e.g atm-ocean, atm-surf-subsurf, sat-unsat),

(2) subdivision of a domain (e.g stratified flow, stratigraphy),

(3) different processes (e.g. precip, snowmelt, infil, seepage, ET)

A good example is a ***distributed hydrologic model***.

Nested = Usually a high-resolution (and maybe 3D) model that is embedded within (and may be driven by) a lower-resolution model. (e.g. regional winds/waves driving coastal currents, or a 3D channel flow model within a landscape model)

Boundary-coupled = Model coupling across a natural (possibly moving) boundary, such as a coastline. Usually fluxes must be shared across the boundary.