Technical Overview of the Community Surface Dynamics Modeling System

Scott Peckham

Chesapeake Focus Research Group Meeting

April 3, 2009. Annapolis, MD.

CSDMS

Community Surface Dynamics Modeling System

Two Powerful Tools for Component-Based Modeling

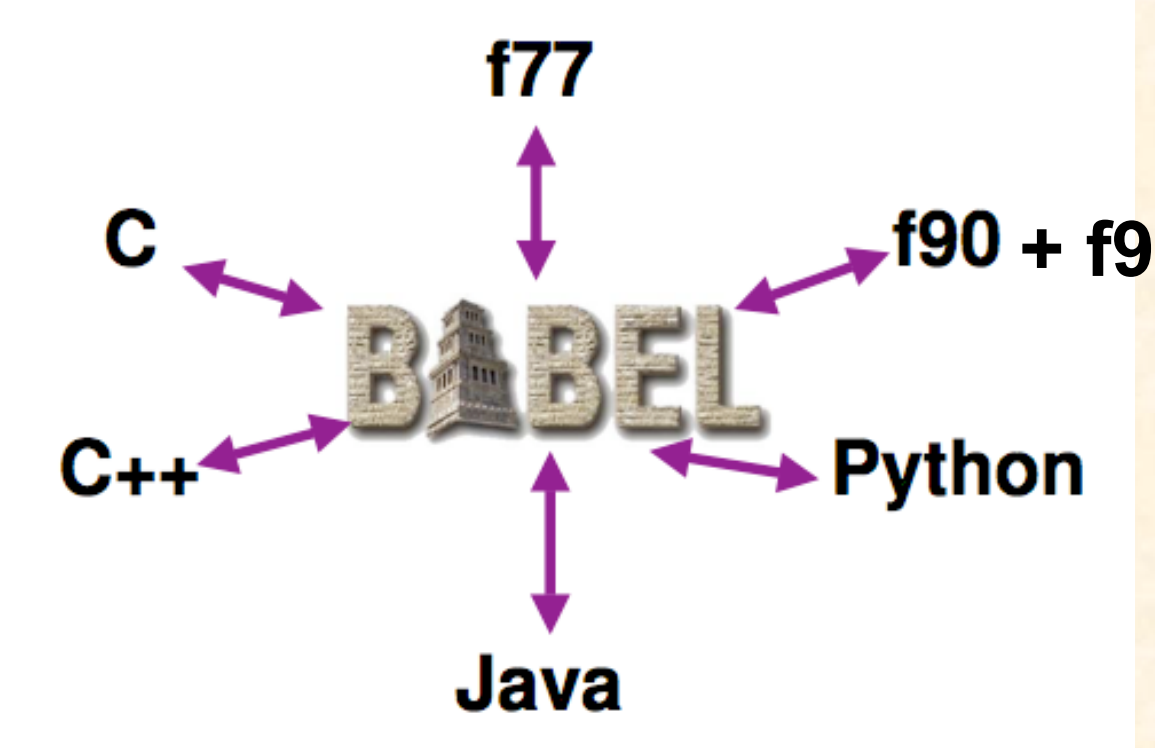
A ***component architecture standard*** (or framework standard) that supports high-performance, scientific computing.

Provides a language-interoperability tool called ***Babel***, RPC support, a component project management tool called ***Bocca***, and a graphical interface for connecting components within the ***Ccaffeine*** framework.

A ***component interface standard*** designed for numerical models of the type where arrays of values march forward in time.

Provides tools for passing data between models that may use ***different computational grids***, ***different dimensionality***, units, etc.

CCA's Babel Language Tool



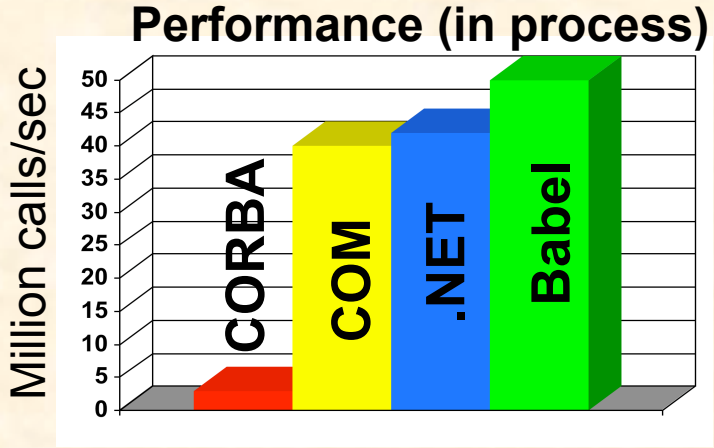
Language interoperability is a powerful feature of the CCA framework. Components written in different languages can be rapidly linked in HPC applications with hardly any performance cost. This allows us to "shop" for open-source solutions (e.g. libraries), gives us access to both procedural and object-oriented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

CCA's Babel Language Tool

is Middleware for HPC

2006

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



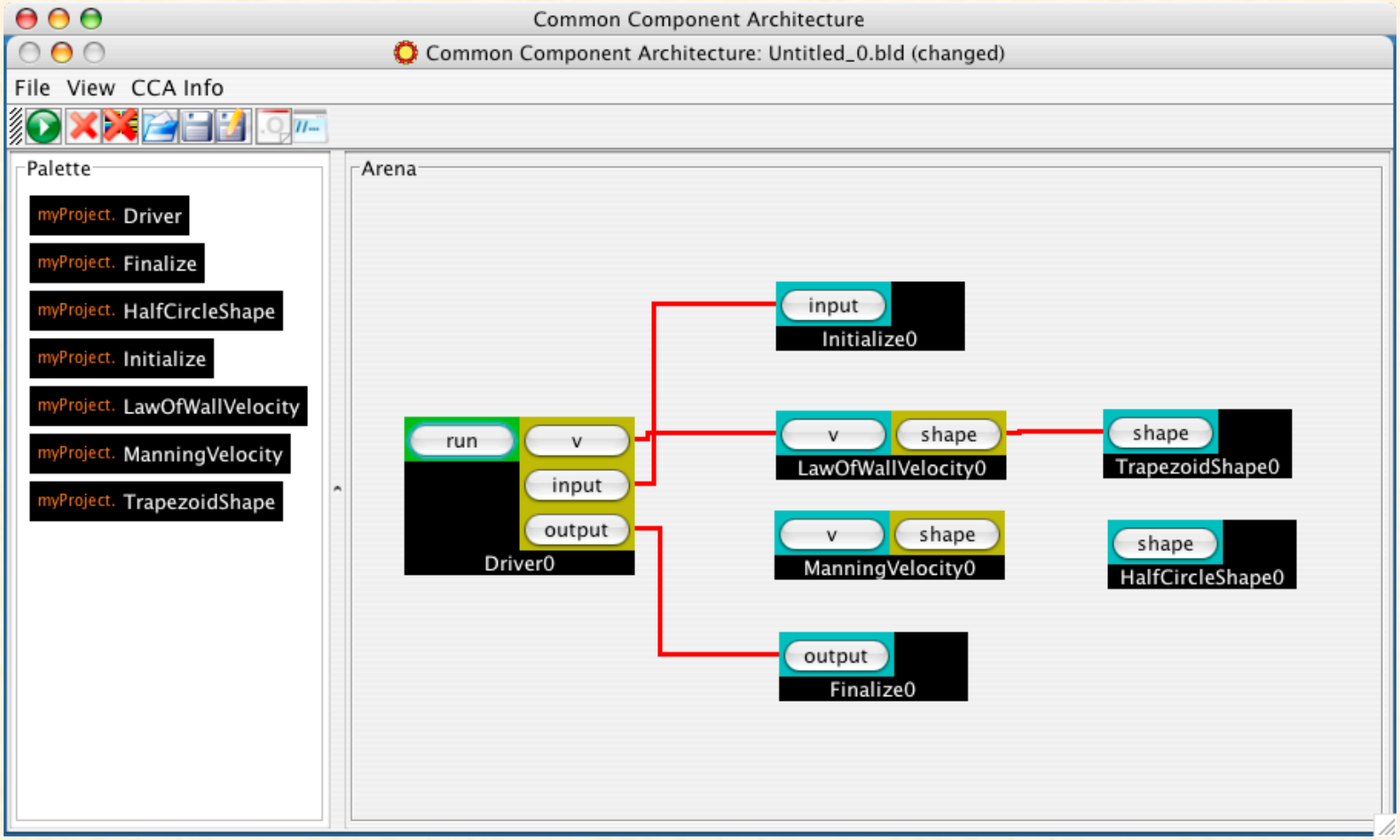
	CORBA	COM	.NET	Babel
BlueGene, Cray, Linux, AIX, & OS X	No	No	No	Yes*
Fortran	No	Limited	Limited	Yes
Multi-Dim Arrays	No	No	No	Yes
Complex Numbers	No	No	No	Yes
Licensing	Vendor Specific	Closed Source	Closed Source	Open Source

CCA's Bocca Project Mgmt. Tool

```
#! /bin/bash
#  Use BOCCA to create a CCA test project.
#  October 23, 2007.   S.D. Peckham
#-----------------------
#  Set necessary paths
#-----------------------
source $HOME/.bashrc
echo "========================================="
echo "  Building example CCA project with BOCCA  "
echo "========================================="

#--------------------------------------
#  Create a new project with BOCCA and
#     Python as the default language
#--------------------------------------
cd $HOME/Desktop
mkdir cca_ex2;  cd cca_ex2
bocca create project myProject --language=python
cd myProject

#--------------------------------
#  Create some ports with BOCCA
#--------------------------------
bocca create port InputPort
bocca create port vPort
bocca create port ChannelShapePort
bocca create port OutputPort

#---------------------------------------
#  Create a Driver component with BOCCA
#---------------------------------------
bocca create component Driver \
            --provides=gov.cca.ports.GoPort:run \
            --uses=InputPort:input \
            --uses=vPort:v \
            --uses=OutputPort:output
```

```
#---------------------------------
#  Create an Initialize component
#---------------------------------
bocca create component Initialize \
            --provides=InputPort:input

#-----------------------------------------------
#  Create two components that compute velocity
#-----------------------------------------------
bocca create component ManningVelocity \
            --provides=vPort:v \
            --uses=ChannelShapePort:shape
bocca create component LawOfWallVelocity \
            --provides=vPort:v \
            --uses=ChannelShapePort:shape

#-------------------------------------------------
#  Create some channel cross-section components
#-------------------------------------------------
bocca create component TrapezoidShape \
            --provides=ChannelShapePort:shape
bocca create component HalfCircleShape \
            --provides=ChannelShapePort:shape

#-----------------------------------
#  Create a Finalize component
#-----------------------------------
bocca create component Finalize \
            --provides=OutputPort:output

#-----------------------------------
#  Configure and make the new project
#-----------------------------------
./configure; make
```

CCA's 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.

"Underemployed" Models

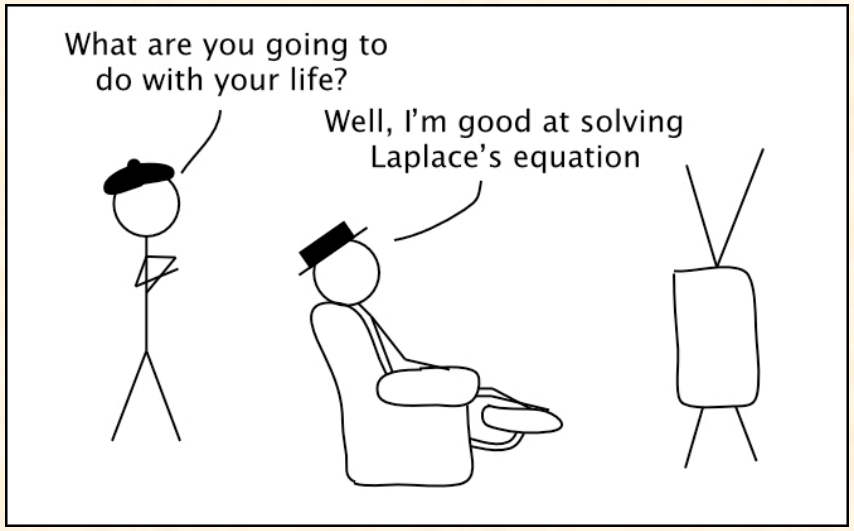
Many surface dynamics models have been developed by our community but they tend to be underutilized. They differ in a number of ways, which makes it difficult for them to be linked together or used interchangeably.

For example, they may:

- be written in ***different languages***
- use ***different grids*** (triangles, rectangles)
- have ***different dimensionality*** (1D, 2D, 3D)
- use ***different formats*** for input/output

The idea behind ***component-based*** modeling is to transform or wrap these models so that they can be used in a ***plug-and-play*** manner.

But how do we deal with these differences?



Components are like people, in that they:

- have particular sets of skills or abilities
- speak some language
- communicate with other components
- work together to solve problems
- make requests of one another
- access and/or require certain services
- need to be "connected" to work together

Recruiting Models to CSDMS

CSDMS seeks to bring open-source models together and to repackage them as components that can be easily reused and connected to other components in order to solve a broad range of surface dynamics problems.

CSDMS requests that code contributors **make a few, relatively small changes to their model's structure** so that we can more easily provide it with a standard interface.

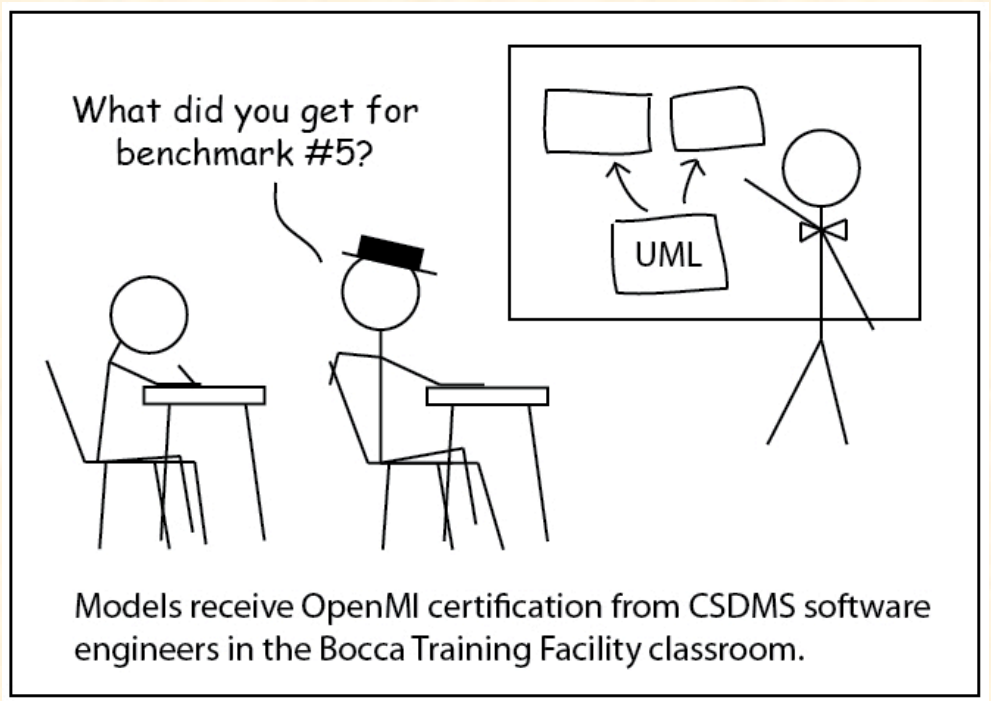
CSDMS also **requires that contributed models can be compiled with a standard, open-source compiler**. We don't have the resources to support all compilers.

Training Models with Bocca

Bocca is a tool in the CCA tool chain that the software engineers can use in order to ***prepare components*** for deployment in a CCA framework.

This preparation has mainly to do with providing components with standard interfaces (e.g. OpenMI) so they are interchangeable and can work together.

Bocca relies on another CCA tool called Babel to ***create language bindings*** that are necessary so that components written in different languages can communicate or call one another.



Models receive OpenMI certification from CSDMS software engineers in the Bocca Training Facility classroom.

The Basic "IRF" Interface

In the context of componentized software, **an *interface* is a named set of member functions** that provide a caller with access to its capabilities. (That is, names and data types of all arguments & return values are completely specified.

A basic "IRF interface" is something that virtually all model coupling efforts have in common (e.g. ESMF, OMS and OpenMI). IRF stands for "Initialize, Run_Step, Finalize". **We want contributors to provide this interface.**

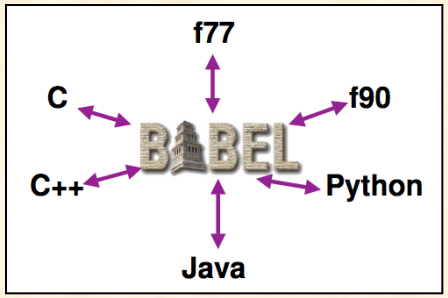
```
Run_Model()
    Initialize()
    while not(DONE):  Update()
    Finalize()
```

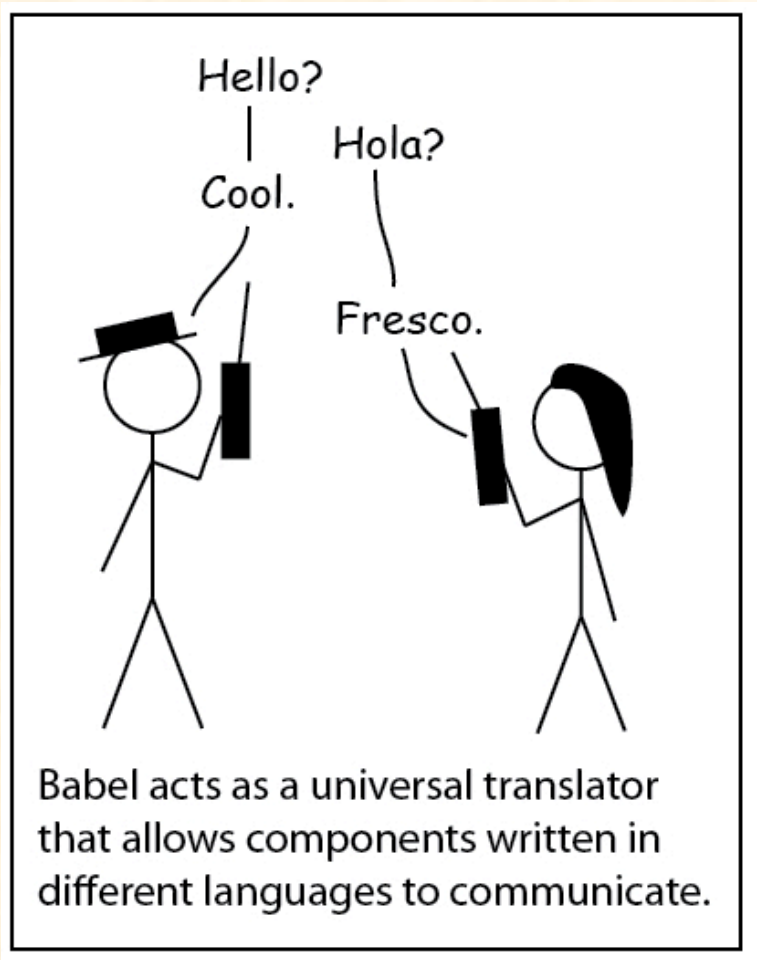
Initialize() => Open files, read/compute initial values, allocate memory, etc.
Run_step() or Update() => Update all of the computed values (one step forward)
Can call other functions, e.g. Update_Q, Update_v.
Finalize() => Close files, print report, free memory.

Helping Components Communicate

Babel is a powerful tool that can provide ***language interoperability*** for components in a CCA framework. This means that there is no need to convert all of our models to a common language. Contributors can keep working in whatever language they want.

Components written in different languages can be rapidly linked in HPC applications with hardly any performance cost. This also allows us to "shop" for open-source solutions (e.g. libraries), lets us mix procedural and object-oriented strategies, and allows us to add graphics & GUIs.





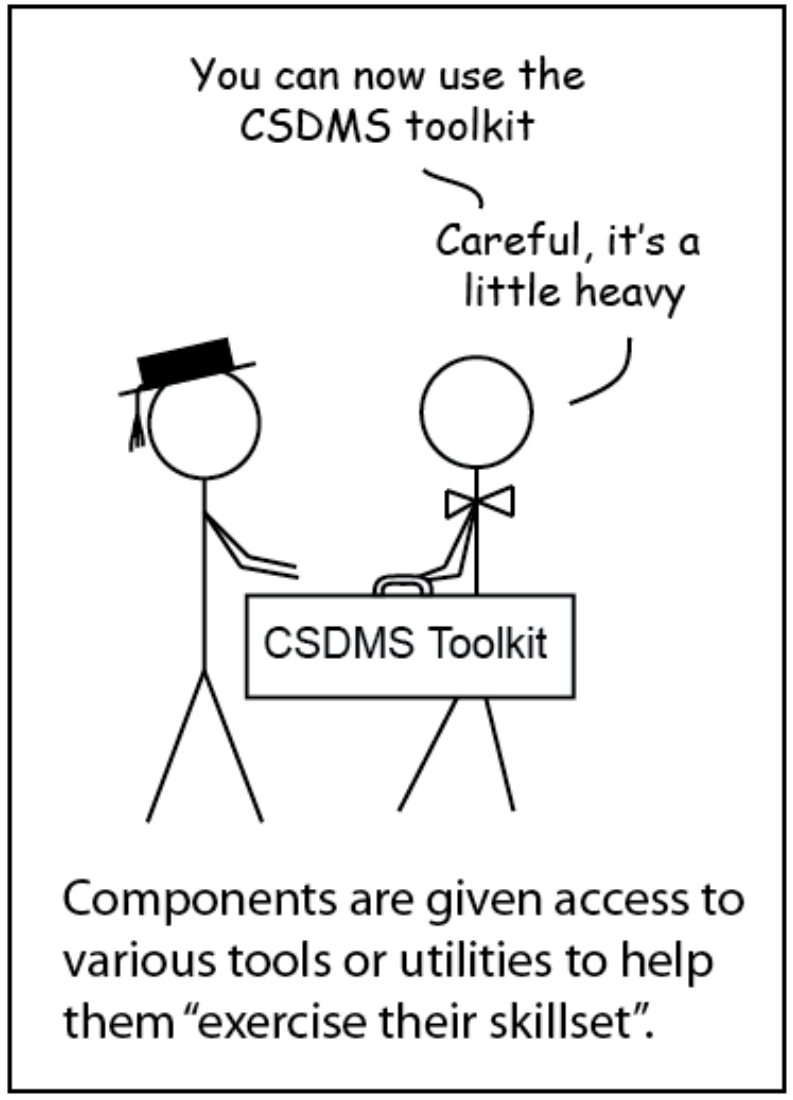
Babel acts as a universal translator that allows components written in different languages to communicate.

Providing Components with Tools

Components may also be provided with "toolkits" or utilities that are needed in order to "***exercise their skillset***."

This is similar to a licensed electrician or doctor needing a tool bag that contains the "tools of their trade."

An interface standard, like OpenMI, may include a ***software development kit*** or ***SDK*** that performs the low-level tasks that are necessary in order to provide a component with an implementation of that interface.

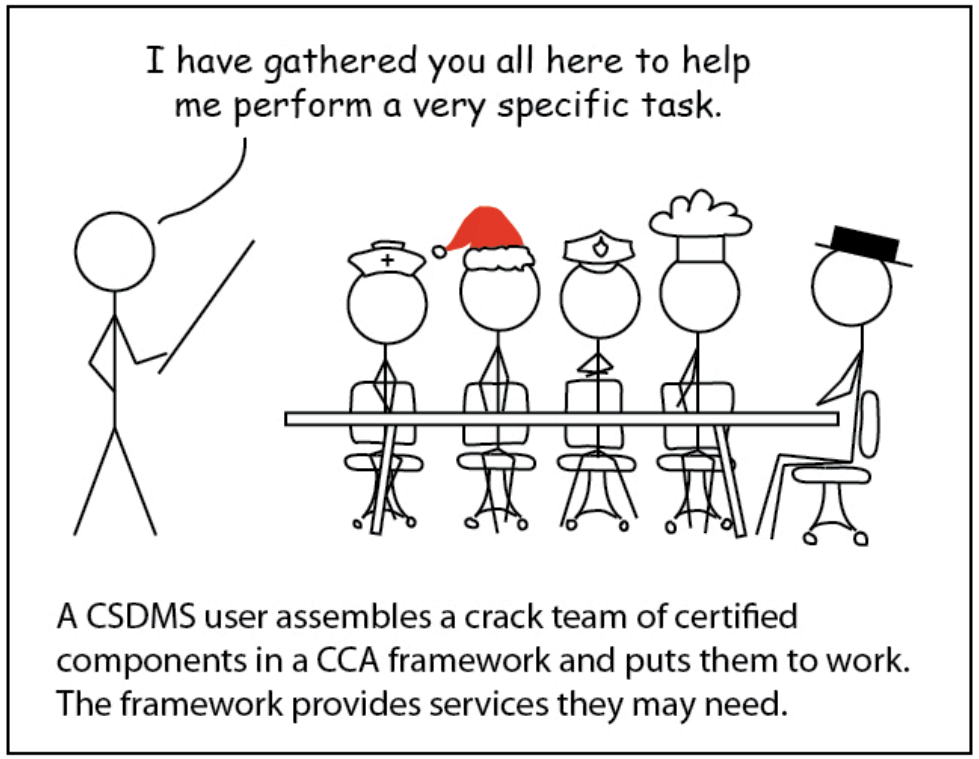


Deploying Models in a CCA Framework

The main difference between subroutines and components is that components are deployed and connected within a ***framework***.

If components are people, then a framework is a venue where people work together on a common goal, such as a boardroom, concert hall, sports arena, battlefield or construction site.

The framework provides a place to work but also typically provides a set of ***services*** that are accessible to every component. It may also provide ***language interoperability*** with a tool like Babel.



A CSDMS user assembles a crack team of certified components in a CCA framework and puts them to work. The framework provides services they may need.

Framework services are analogous to services that the venue provides, such as wireless internet, electricity, projectors, telephones, heat, light and catering.

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 a basic ***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

Summary

CSDMS employs a component-based modeling approach.

CSDMS authors contribute models with an “IRF interface”, and in a Babel supported language (C, C++, Fortran, Python, Java)

CSDMS staff assists with converting models to plug-and-play components that have a standardized OpenMI interface.

CSDMS users can then assemble new models from these components in a CCA-compliant framework. The Ccaffeine GUI can be used locally to assemble a model and then the model can be run remotely on the CSDMS supercomputer.

CCA Babel takes care of language issues, while the OpenMI SDK takes care of other differences between models such as grid type and dimensionality.

This CSDMS approach has been demonstrated with prototypes.