CCA
Common Component Architecture

Introduction to the Ccaffeine Framework

CCA Forum Tutorial Working Group
http://www.cca-forum.org/tutorials/

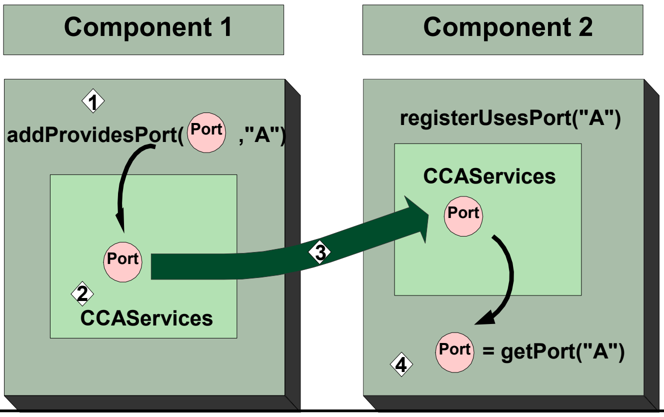
CCA
Common Component Architecture

Outline

- What is a CCA Framework and what is Ccaffeine?
- How can I slip my own component into Ccaffeine?
- How do I run Ccaffeine?
- Live Demo – does it work?

CCA What CCA compliant framework is expected to do …

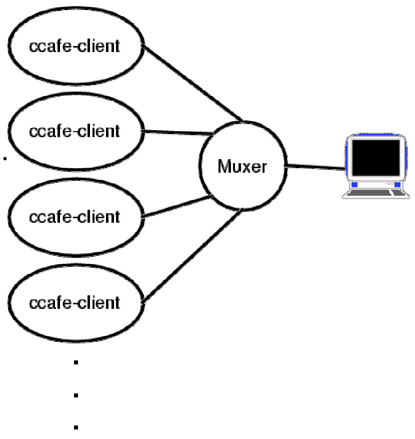
- Exchange interfaces among components without one component needing to know more about the other than the interface itself.



Interactive Parallel Components: what Ccaffeine does

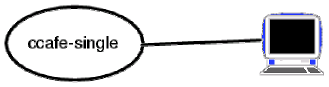
- Executable ccafe-client:
  - PVM, MPI, or whatever is used for communication between clients.
  - Muxer enforces “single process image” of SPMD parallel computing.
- HOWTO: http://www.cca-forum.org/ccafe/
  - Build Ccaffeine
  - Run Ccaffeine

http://www.cca-forum.org/ccafe/

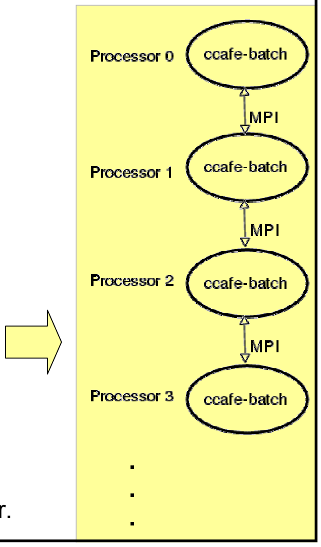


Ccaffeine comes in two other flavors* and a GUI.

- Single process executable: ccafe-single
  - really useful for debugging



- Batch executable: ccafe-batch
  - when all you want to do is run it.



*flavor: same executable, different name and behavior.

How to build Ccaffeine

- Have a look at http://www.cca-forum.org/ccafe
  - Obtain the required packages
    - Ccaffeine tar ball download
    - gcc (2.95.3, 2.96, *not* 3.x)
    - Java (>jdk1.2)
    - BLAS, LAPACK (any recent)
    - BOOST headers
    - Babel (0.7.0 *only*)
    - Ruby (any recent, if you have Linux, probably there now)

CCA
Common Component Architecture

How to build Ccaffeine (cont'd)

- Untar Ccaffeine-xxx.tgz in build dir
  - 3 directories appear cca-spec-babel (*the* spec), cca-spec-classic (old C++ spec), dccafe
- Run configure
  - If confused type "configure –help"

```
(cd ./cca-spec-babel; configure --with-babel=/usr/local/babel \
--with-jdk12=/usr/local/java;make)

(cd ./cca-spec-classic;configure;make)

(cd ./dccafe; ./configure  --with-cca-babel=`pwd`/../cca-spec-babel \
--with-cca-classic=`pwd`/../cca-spec-classic \
--with-mpi=/usr/local/mpich --with-jdk12=/usr/local/java \
--with-lapack=/home/rob/cca/dccafe/../LAPACK/liblapack.a \
--with-blas=/home/rob/cca/dccafe/../LAPACK/libblas.a; make)
```

CCA
Common Component Architecture

Ccaffeine build (cont'd)

- The Ccaffeine make will take ~5-10 min.
- Look in:
  http://www.cca-forum.org/ccafe/build-log.html
  for a complete listing from Rob's laptop.

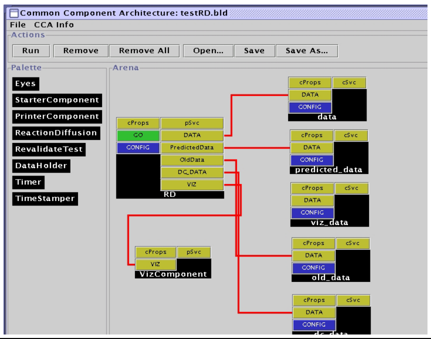
If successful you should get:

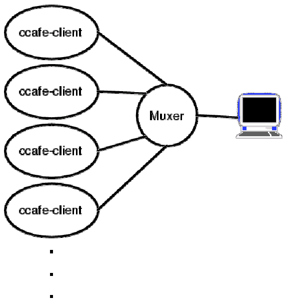
```
=====================================================================
    Testing the Ccaffeine build ...
    didn't crash or hang up early ...  looks like it is working.
    done with Ccaffeine tests.
=====================================================================
```

CCA
Common Component Architecture

How to run Ccaffeine:

- Ccaffeine interactive language: “benSpeak”
  - used to configure batch and interactive sessions.
  - Allows useful “defaults.”
  - Allows the GUI to talk over a socket.





CCA
Common Component Architecture

Ccaffeine scripting language is for those who have grown tired of the GUI

- look in:
  http://www.cca-forum.org/ccafe/ccafe-man/Ccafe_Manual.html
  for all the commands.
- The GUI is just a pretty front end that speaks this scripting language to the backend.

You can talk directly to Ccaffeine by typing:

```
prompt> ccafe-single
MPI_Init called in CmdLineClientMain.cxx
my rank: 0, my pid: 25989
... (output cruft deleted)
cca>help
(complete listing of commands and what they do)
```

Quick run-through of the Ccaffeine scripting language

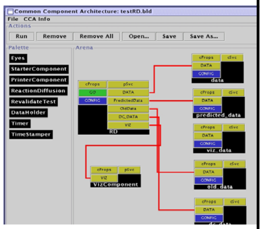
- scripting language does everything that the GUI does.
- Warning: there are two of files that Ccaffeine uses:
  - "rc" and script files for building and running apps
  - GUI ".bld" files that are state saved by the Ccaffiene GUI.

These are not the same and will give, sometimes spectacular, undefined behavior.

Magic number and repository function: the top of the script

- Must tell the framework where the components are ("path") and which ones you want loaded into the "pallet".

```
#!ccaffeine bootstrap file.
# ------- don't change anything ABOVE this line.-------------
# where to find components:
path set /home/rob/cca/component
# load components into the "pallet"
repository get functions.PiFunction
repository get integrators.MonteCarloIntegrator
repository get integrators.MidPointIntegrator
repository get integrators.ParallelIntegrator
repository get randomgen.RandRandomGenerator
repository get tutorial.driver
```

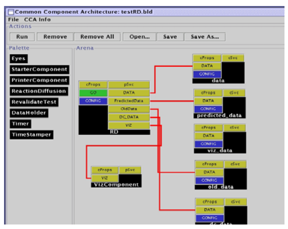


- At this point no components are instantiated, but are simply known to the system.

Now start instantiating the components that will form your application

- Use the “create” function to make an instance of a component and name it.
  - first arg is the class name of the component and the second is the instance name you want it to have:

```
# Instantiate and name components that have been made
# known to the framework
create randomgen.RandRandomGenerator rand
# f(x) = 4.0/(1 + x^2)
create functions.PiFunction function
create tutorial.Driver driver
```



Connect the components to form a complete application

- Connect takes 4 arguments, all of them are instance names of components or ports. In order they are:
  1. Using component instance name (named in “create”).
  2. Uses port instance name (name given to it by the component)
  3. Providing component instance name.
  4. Provides port instance name.
- Script from our example code:

```
# Connect uses and provides ports
connect integrator FunctionPort function FunctionPort
connect integrator RandomGeneratorPort rand RandomGeneratorPort
connect driver IntegratorPort integrator IntegratorPort
```

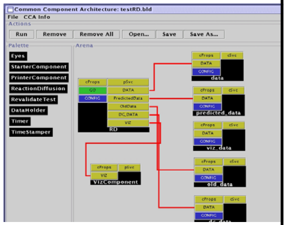
CCA
Common Component Architecture

Time to see if it works: the “go” command

- The “go” command takes a component instance and a port instance name as an argument
  – only the named port on the named component are **`go()`**’ed:

```
# Good to go()
go driver GoPort
```

- At this point Ccaffeine gets completely out of the way.
  – So much so that it will not respond until (or if) your application returns from the invocation of the “go()” method.
  – There **is** only one thread of control.



CCA
Common Component Architecture

CCA is working on a component delivery specification, until then Ccaffeine has some specific req’ts

- “.cca” file describes what the format of the component is: “Babel”, or old-style “Classic.”
- Component wrapper class
  – introduces to the framework one or more components
  – contained in the “.so” file with the component(s).
  – will go away for Babel components.

Example “.cca” file: MonteCarloIntegrator in integrators.cca

- Ccaffeine-specific file specifying the name of the dynamic library and creation method for each component

```
!date=Thu Aug 15 14:53:23 CDT 2002
!location=
!componentType=babel
libIntegrator-component-c++.so
create_MonteCarloIntegrator integrators.MonteCarloIntegrator
```

Component type: “babel” or “classic” (C++)

“.so” Library

C wrapper function name

Component name

Wrapper C functions

- auto-gen the wrapper C code file:
  - “genDL” scripts provided by Ccaffeine.
  - genDLWrapperStrict to generate the “.cca” file.
  - usage: genDLWrapper <component class name>
- creates the appropriate symbols to be included in the “.so” file so that Ccaffeine can find and instantiate the component.
- In the case of Babel components this step is unnecessary and is soon to be removed.

What you are able to do now that you couldn't before …

- Run on parallel cluster or proprietary machine with CCA components that you didn't write.
  - Steve Jobs: "the best software is software I didn't have to write" –not that he actually ever did.
- Develop incrementally & interactively in serial and *parallel*.
  - Detach, go have lunch and reattach.