CCA
Common Component Architecture

# Introduction to the Ccaffeine Framework

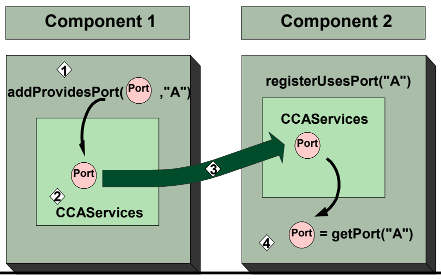
**CCA Forum Tutorial Working Group**
http://www.cca-forum.org/tutorials/
*tutorial-wg@cca-forum.org*

## 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 – how does it work?

## CCA What CCA compliant framework is expected to do …

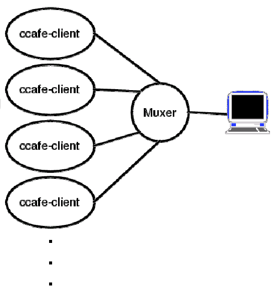
- Exchange interfaces among components without one component needing to know more about the other than the interface itself



## 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
- How To:
  - Build Ccaffeine
  - Run Ccaffeine

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



## How to build Ccaffeine

- Have a look at http://www.cca-forum.org/ccafe
  1. Obtain the required packages
     - gcc (http://gcc.gnu.org)
     - Java (>jdk1.2) (http://java.sun.com)
     - MPI (http://www.mcs.anl.gov/mpi/mpich)
     - BOOST headers (http://www.boost.org)
     - Babel (http://www.llnl.gov/casc/components/babel.html)
     - Ccaffeine tar ball download (or rpm)
     - Optional software
       - Fortran 77 and 90 compilers
       - Ruby
       - Python 2.x
  2. Install prerequisites

## 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”; example options:

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

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

(cd ./dccafe; ./configure  --with-cca-babel=`pwd`/../cca-spec-babel \
--with-cca-classic=`pwd`/../cca-spec-classic –with-babel=/usr/local/babel-0.8.4 \
--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; make install)
```

## Ccaffeine build (cont’d)

- Example output at “make install” completion:

```
=====================================================================
                  Testing the Ccaffeine build ...
proceeding with env vars:
# LD_LIBRARY_PATH=/home/norris/cca/dccafe/cxx/dc/babel/babel-
   cca/server:/home/software/mpich-1.2.5-
   ifc/lib/shared:/home/norris/cca/babel-
   0.8.4/lib:/usr/local/lib/python2.2/config:/usr/local/intel/compiler70/
   ia32/lib:/usr/local/lib:/usr/local/lib
# SIDL_DLL_PATH=/home/norris/cca/dccafe/lib
didn't crash or hang up early ...  looks like it is working.
Looks like CLASSIC dccafe is working.
Looks like BABEL dccafe is working.
done with Ccaffeine tests.
simpleTests: output is in
   /home/norris/cca/dccafe/simpleTests.out.XXXAL8Cmk.
=====================================================================
```

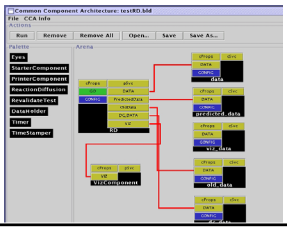
Note: depending on environment settings, sometimes the simple tests may fail but you may still have a functional framework.

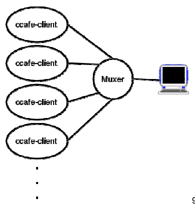
## Running Ccaffeine

- Framework needs to be told:
  - Where to find components
  - Which components to instantiate
  - Which **uses** port gets connected to which **provides** port
  - Which **go** port sets the application in motion
- User-Ccaffeine interaction techniques:
  - GUI interface (with some Ccaffeine scripting help)
  - Pure Ccaffeine scripting (useful in batch mode)
  - Python component driver (with some Ccaffeine scripting help)

## How to run Ccaffeine:

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



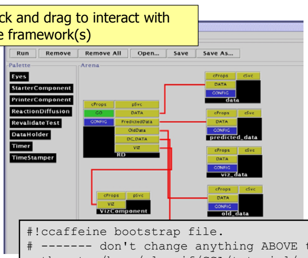


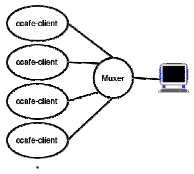
## The Ccaffeine GUI

- Java front end to one (or more) *framework* instances running in the background
- Events propagated to all frameworks through a *muxer*
- Framework(s) still need Ccaffeine script to know about available components
- GUI used to instantiate, connect, and configure components (and to launch the whole application as well)
- Usage modes:
  - Compose and launch application from scratch (graphically).
  - Load *pre-composed* applications (the .bld files)

## The GUI

Click and drag to interact with the framework(s)





Component paths and types needed by the framework(s) (the .rc files)

```
#!ccaffeine bootstrap file.
# ------- don't change anything ABOVE this line.-------------
path set   /home/elwasif/CCA/tutorial/src/sidl/random-component-c++
path append /home/elwasif/CCA/tutorial/src/sidl/function-component-c++
path append /home/elwasif/CCA/tutorial/src/sidl/integrator-component-c++
path append /home/elwasif/CCA/tutorial/src/sidl/driver-component-c++
repository get randomgen.RandRandomGenerator
repository get functions.LinearFunction
repository get functions.PiFunction
repository get functions.NonlinearFunction
```

SIDL_DLL_PATH environment variable also used for locating component shared libraries!

## The Command Line Way: Using Ccaffeine Scripting

- Simple scripting “language” to talk to the framework.
- For the full list of commands:

```
UNIX>ccafe-single ↵
    cca> help ↵
```

- Some commands:

```
–  path set  <initial path to components>
–  path append <directory containing component code>
–  repository get <component class>
–  instantiate <component class> <component name>
–  connect <use component name> <use port name> \
           <provide component name> <provide port name>
–  go <component name> <Go port name>
–  bye
```

## Ccaffeine scripting language for batch use

- Two modes of execution:
  - ccafe-single : uniprocessor, interactive, no MPI
  - ccafe-batch or ccafe-client: parallel jobs, GUI
- Refer to http://www.cca-forum.org/ccafe/ccafe-man/Ccafe_Manual.html for more detailed description of the commands

You can run Ccaffeine interactively 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 files that Ccaffeine uses to locate and load component libraries:
  - "rc" and script files for building and running apps
  - GUI ".bld" files that store state saved by the Ccaffeine GUI

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

## Example: example1_rc

```
#!ccaffeine bootstrap file.
# ------- don't change anything ABOVE this

path set  /home/elwasif/CCA/tutorial/random-component-c++
path append /home/elwasif/CCA/tutorial/function-component-c++
path append /home/elwasif/CCA/tutorial/integrator-component-c++
path append /home/elwasif/CCA/tutorial/driver-component-c++
# 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
```

SIDL_DLL_PATH environment variable also used for locating component shared libraries!

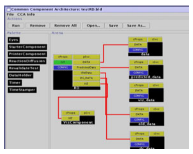
Component classes/types

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

## Example (cont.): Instantiation

```
create randomgen.RandRandomGenerator rand
create functions.PiFunction function
create integrators.MonteCarloIntegrator integrator
create tutorial.Driver driver
```

Component intances names



## Example (cont.): Connection

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

"Uses" ports names

"Provides" ports names

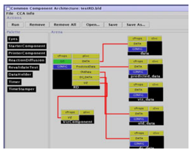
## Example (cont.): Application Launch

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

Provided Go port name

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



## The third way: Using CCA BuilderService

- Deficiencies of Ccaffeine Scripting
  - Non "standard"
  - No error checking !!!!
- Solution: Use a more "complete" scripting language, e.g. Python
- Why Python?? Supported By Babel
- Strategy:
  - Use a Python "mega driver" to assemble the application
  - Talk to the framework through ***`BuilderService`*** interface
  - Still need snippets of Ccafeine scripting to set paths, instantiate python driver, and launch it

## The BuilderService Port

- "*Provided*" by the Framework, "*used"* by any component
- Major methods:
  - **`createInstance(instanceName, className, properties)`**
  - **`connect(userID, usePortName, providerID, providPortName)`**
  - See file cca.sidl for complete interface.
- Many more methods
- Can be "***used***" from any language, Python just more convenient
- See ***`driver-python`*** for details

## Component discovery and instantiation

- CCA is working on an XML component delivery specification, until then Ccaffeine has some specific requirements
- ".cca" file describes what the *type* of the component is: e.g., "babel" or "classic"(Pre-Babel / C++ only binding).

```
!date=Thu Jul 3 14:53:23 EDT 2003
!location=
!componentType=babel
dummy_libIntegrator-component-c++.so
dummy_create_MonteCarloIntegrator integrators.MonteCarloIntegrator
```

Component type: "babel" or "classic" (C++)

.so library containing component (**pre-Babel**)

Component creation function; (**pre-Babel**)

Component type(class); (**pre-Babel**)

## Showing How it All Works

**The Scripts**