# A Component Based Services Architecture for Building Distributed Applications

**Randall Bramley**, Kenneth Chiu,
Shridhar Diwan, Dennis Gannon,
Madhusudhan Govindaraju, Marie Ma,
Nirmal Mukhi, Benjamin Temko,
Madhuri Yechuri

Extreme Computing Lab, Indiana University

1

## Outline

- Trends in CS&E
- Component frameworks
- Brief history
- Features of CCAT
- Recent developments in CCAT

## Trends in CS&E

- Size, complexity, sophistication of apps and libraries is growing exponentially
- Scale and degree of heterogenity is growing
  - Languages
  - Real-time instrument access
  - Data bases, data mining engines, integrated visualization
- Collaborative, multidisciplinary teams
- Trend to both OSS and secret/secure codes and data
- Severe personnel shortage

## Component Features

- Now standard paradigm in industry and commerce
  - COM/DCOM, Java Beans, Enterprise Java Beans, Corba
- Modules distributed across networks
- Well-defined interfaces, independent of language
- Composable dynamically without recompilation to create applications
- Flock of CSE component systems being developed:
  - SciRun (Utah), WebFlow (Fox), NetSolve (UTK), Legion (UVA), and many national lab efforts.
- Need not be software ...

## X-Ray Crystallography Lab

## Why not use CORBA/DCOM/Beans ?

- **Promptness**: we needed a component framework four years ago
- **Efficiency**: a ruling principle of CS&E research apps
- **Parallelism**: need to connect components consisting of incommensurate numbers of MPI processes
- **Simplicity**: target the minimal specs possible
- Nevertheless it is important to interoperate with commercial systems

## CCAT History

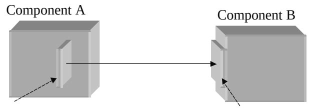
- 1995: Linear System Analyzer: used Nexus + HPC++ for run-time system and data flow model
- 1996-1997 Component Architecture Toolkit: more generic in application areas; still data flow
- 1998: Industrial finite element SC98 demo, with multiple CAVEs/I-Desks for visualization.
- 1998: DoE Common Component Architecture Forum specifications released - start of CCAT.

## CCAT Features

- Service-based architecture - where each service is a CCA-compliant component
- Multiple user interaction systems (including new “Portals” effort)
- Multiprotocol communications between components

## Common Component Architecture (CCA) Ports

- **Ports**: the public interfaces that a component uses or provides.
- Framework defines a mechanism to link uses ports of one component to the provides ports of another.
- CCA only specifies *port services*: register, access, get info about them.



Uses Port is a call site for an interface to an external component

Provides Port - an interface to a service provided by that component.

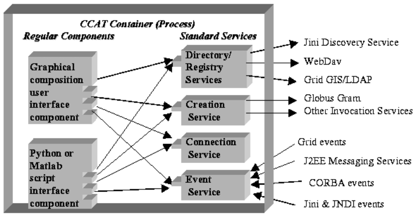
## CCAT Framework Requirements

- A framework must provide other services:
  - Directory Service
    - Locate suitable components
  - Registry Service
    - Locate instantiations of components
  - Creation Service
    - Instantiate a component
  - Connection Service
    - Connect the ports of two running component instances
  - Event Service
    - Publish/subscribe messaging between services and components.

## Services as Components

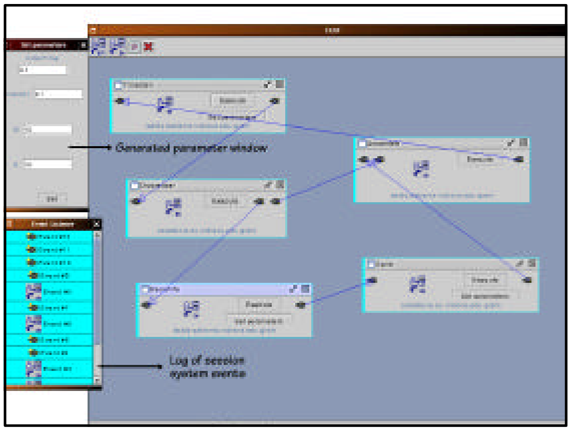
- Each service is a pseudo-component (needs special hooks into CCA core services, bootstrapping)
- E.g.: connection service has port with four methods
  - **Connect** two typed ports
  - **Disconnect**
  - **ExportAs** lets a component export ports of another, so that connection seems to be to first component
  - **ProvideTo** lets a component provide a port to another without registering for the whole CCAT app to access

## CCAT Framework



## Services as Components

- User interaction system is also modular
  - **Custom built GUI**



## Services as Components

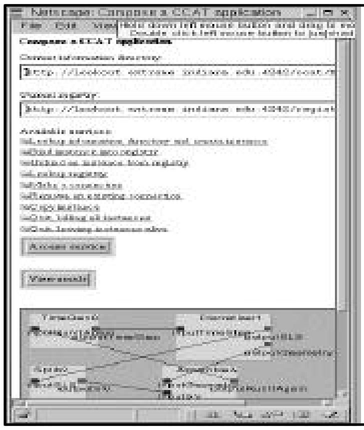
- User interaction system is also modular
  - **Custom built GUI**
  - **Python script**

## Python Example

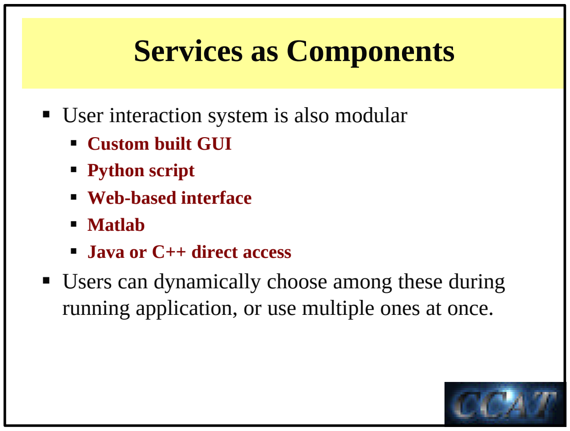
```
import ccat

stringDump = ccat.createComponent('StringDump')
printer = ccat.createComponent('Printer')

ccat.setCreationMechanism(stringDump, 'gram')
ccat.setCreationMechanism(printer, 'gram')

ccat.createInstance(printer)
ccat.createInstance(stringDump)

ccat.connectPorts(stringDump, 'outputString',
                        printer, 'inputString')
```

## Services as Components

- User interaction system is also modular
  - **Custom built GUI**
  - **Python script**
  - **Web-based interface**

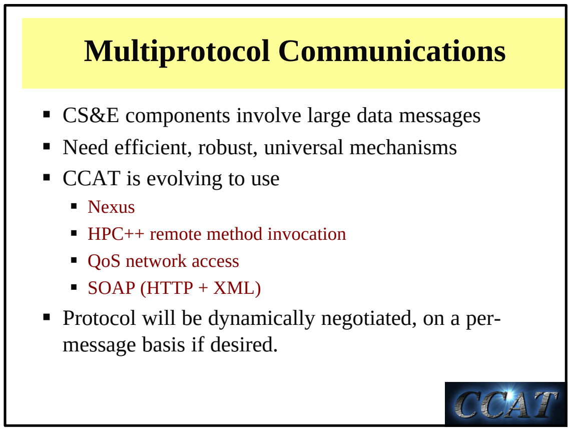

Netscape: Compose a CCAT application
File
Edit
View
Compose a CCAT application
Available services:
Access service
View session

Composer is a CCA component instantiated as a Java servlet.



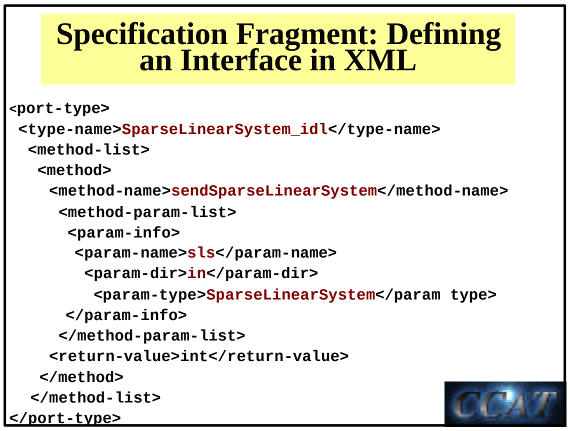
## Services as Components

- User interaction system is also modular
  - **Custom built GUI**
  - **Python script**
  - **Web-based interface**
  - **Matlab**
  - **Java or C++ direct access**
- Users can dynamically choose among these during running application, or use multiple ones at once.



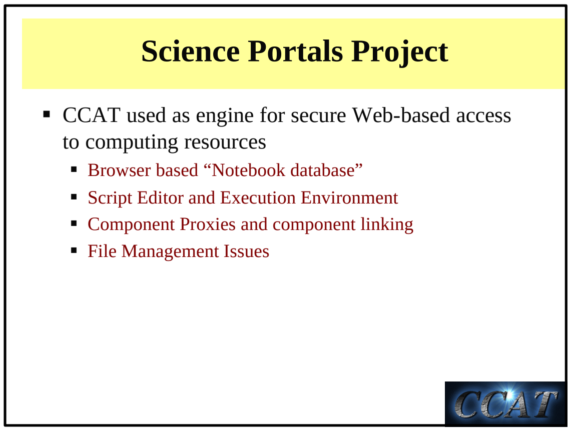
## Multiprotocol Communications

- CS&E components involve large data messages
- Need efficient, robust, universal mechanisms
- CCAT is evolving to use
  - Nexus
  - HPC++ remote method invocation
  - QoS network access
  - SOAP (HTTP + XML)
- Protocol will be dynamically negotiated, on a per-message basis if desired.



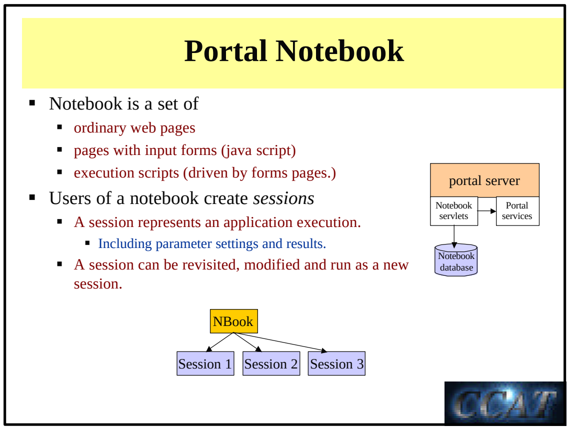
## Specification Fragment: Defining an Interface in XML

```
<port-type>
 <type-name>SparseLinearSystem_idl</type-name>
  <method-list>
   <method>
     <method-name>sendSparseLinearSystem</method-name>
      <method-param-list>
       <param-info>
        <param-name>sls</param-name>
          <param-dir>in</param-dir>
            <param-type>SparseLinearSystem</param type>
       </param-info>
      </method-param-list>
     <return-value>int</return-value>
   </method>
  </method-list>
</port-type>
```



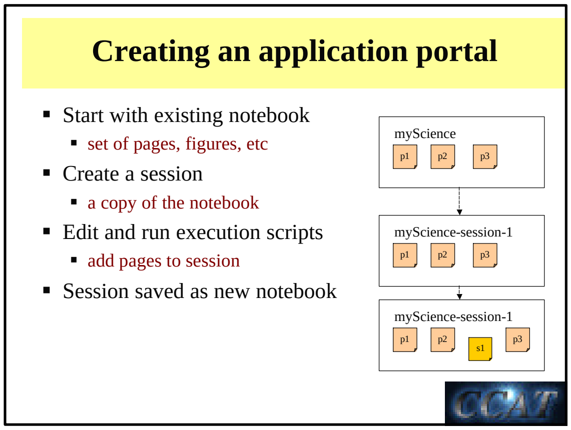
## Science Portals Project

- CCAT used as engine for secure Web-based access to computing resources
  - Browser based "Notebook database"
  - Script Editor and Execution Environment
  - Component Proxies and component linking
  - File Management Issues



## Portal Notebook

- Notebook is a set of
  - ordinary web pages
  - pages with input forms (java script)
  - execution scripts (driven by forms pages.)
- Users of a notebook create *sessions*
  - A session represents an application execution.
    - Including parameter settings and results.
  - A session can be revisited, modified and run as a new session.

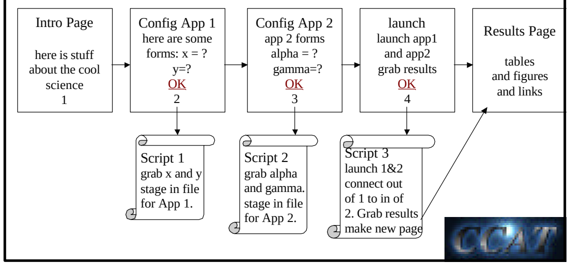


## Creating an application portal

- Start with existing notebook
  - set of pages, figures, etc
- Create a session
  - a copy of the notebook
- Edit and run execution scripts
  - add pages to session
- Session saved as new notebook

## Example.

- A notebook that launches two applications and returns results.



Intro Page

here is stuff about the cool science

1

Config App 1

here are some forms: x = ? y=?

OK

2

Config App 2

app 2 forms alpha = ? gamma=?

OK

3

launch

launch app1 and app2 grab results

OK

4

Results Page

tables and figures and links

Script 1

grab x and y stage in file for App 1.

Script 2

grab alpha and gamma. stage in file for App 2.

Script 3

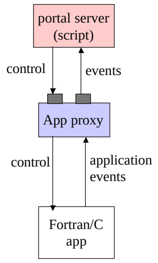
launch 1&2 connect out of 1 to in of 2. Grab results make new page

## Scripting in Notebook

- Notebook has built-in interactive script/forms editor
  - Interactive forms layout and testing.
  - Allows notebook chapter designer :
    - semi-interactive design of application scripts.
    - Easy-to-use forms editor
    - all from standard web browser (no plug-ins)
- Scripting language is JPython
  - gives full access to CCAT, COG, GDK class libraries
  - powerful language with growing popularity in scientific computing.

## Application Components

- Applications run as a stand alone programs
  - reads and writes files
  - may send and receive “event” messages.
- Applications can have an Application Proxy.
  - Provides a component interface to app.
  - Provides sequencing control for IO staging



portal server (script)

control

events

App proxy

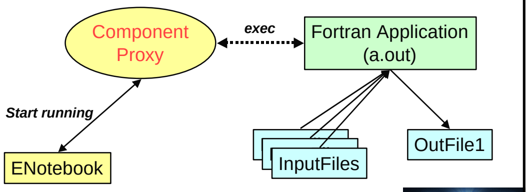
control

application events

Fortran/C app

## Component Proxy/Manager Model

- Application may be untouchable (no source code, etc)
- Idea is to make it appear as a fully-enabled component
- Create a proxy that manages app and framework comm.



Component Proxy

exec

Fortran Application (a.out)

Start running

ENotebook

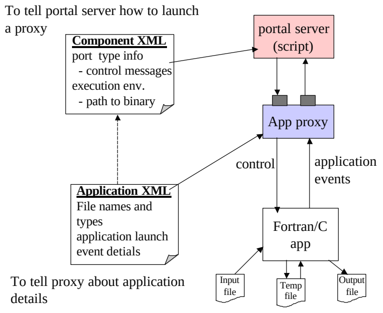
InputFiles

OutFile1

## Component Proxy/Manager Model

- Proxy is responsible for
  - making sure all input files are ready before running
  - event notifications to ENotebook and other interested parties
  - publishing file locations, and moving files
- When output file of one component is needed as input file of another, receiver is responsible for file move.
- Component proxy has user’s Globus certificate of authority and can use gsiftp, gsissh for file transfers, app execution.
- Proxy can (on advice of resource recommender) actually run application on a different machine

## XML encoded application metadata



To tell portal server how to launch a proxy

**Component XML**

port type info

- control messages

execution env.

- path to binary

portal server (script)

App proxy

control

application events

**Application XML**

File names and types

application launch event detials

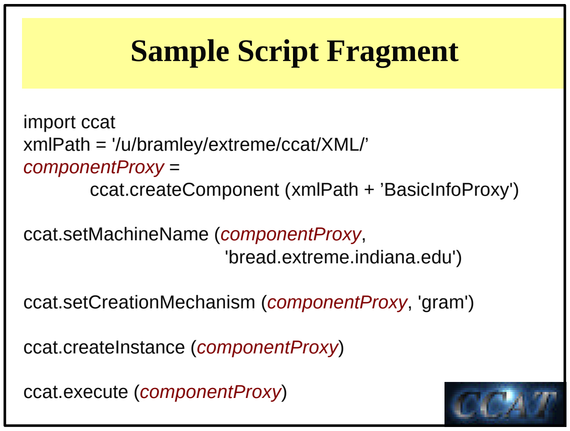
Fortran/C app

To tell proxy about application details

Input file

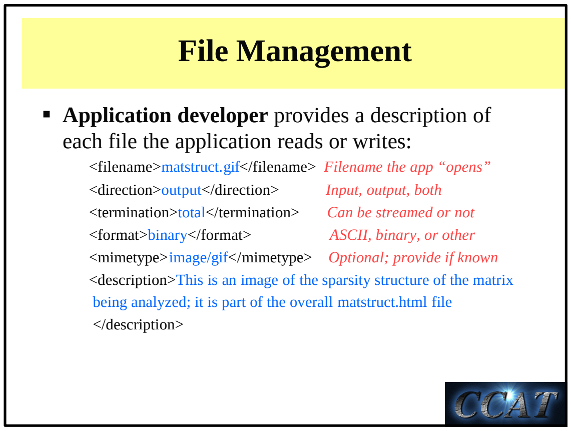
Temp file

Output file



## Sample Script Fragment

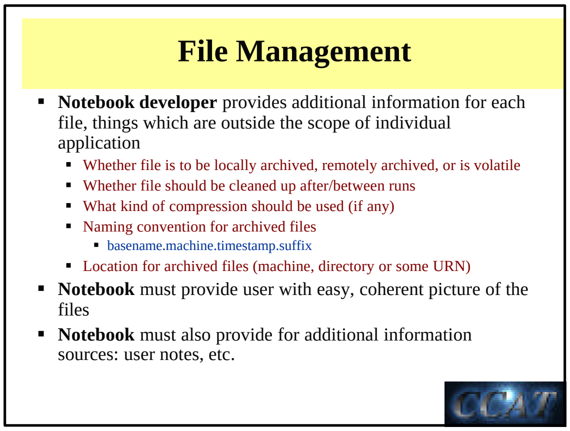
```
import ccat
xmlPath = '/u/bramley/extreme/ccat/XML/'
componentProxy =
          ccat.createComponent (xmlPath + 'BasicInfoProxy')

ccat.setMachineName (componentProxy,
                              'bread.extreme.indiana.edu')

ccat.setCreationMechanism (componentProxy, 'gram')

ccat.createInstance (componentProxy)

ccat.execute (componentProxy)
```



## File Management

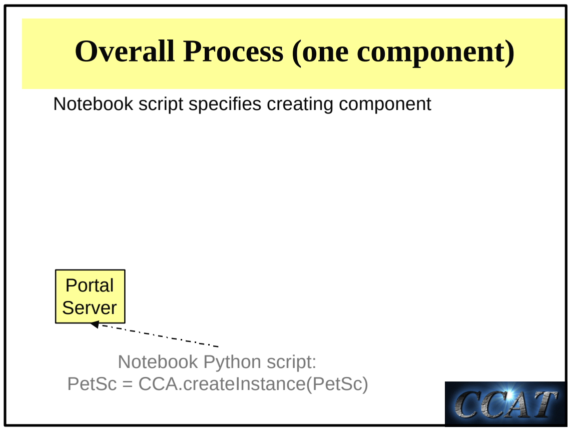
- **Application developer** provides a description of each file the application reads or writes:

```
<filename>matstruct.gif</filename>  Filename the app "opens"
<direction>output</direction>        Input, output, both
<termination>total</termination>     Can be streamed or not
<format>binary</format>              ASCII, binary, or other
<mimetype>image/gif</mimetype>       Optional; provide if known
<description>This is an image of the sparsity structure of the matrix
 being analyzed; it is part of the overall matstruct.html file
 </description>
```



## File Management

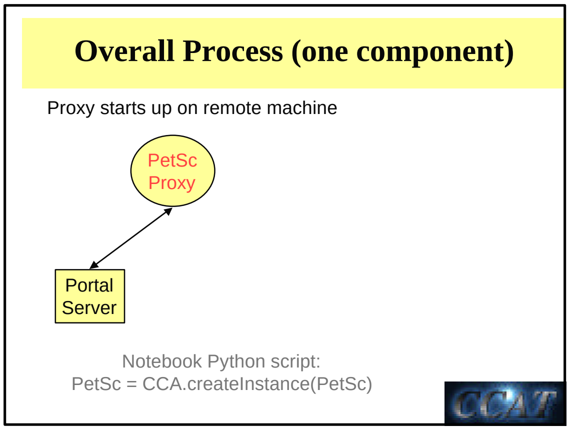
- **Notebook developer** provides additional information for each file, things which are outside the scope of individual application
  - Whether file is to be locally archived, remotely archived, or is volatile
  - Whether file should be cleaned up after/between runs
  - What kind of compression should be used (if any)
  - Naming convention for archived files
    - basename.machine.timestamp.suffix
  - Location for archived files (machine, directory or some URN)
- **Notebook** must provide user with easy, coherent picture of the files
- **Notebook** must also provide for additional information sources: user notes, etc.



## Overall Process (one component)

Notebook script specifies creating component

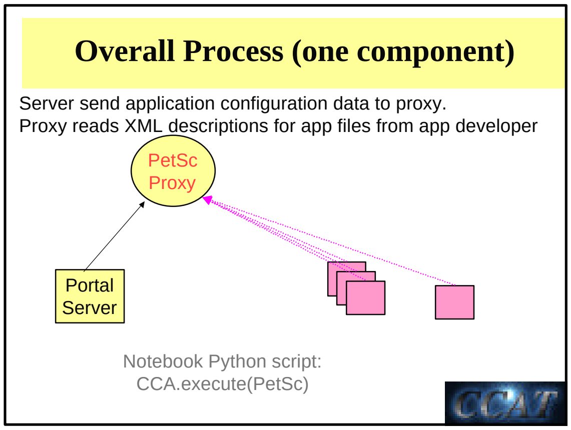
Notebook Python script:
PetSc = CCA.createInstance(PetSc)



## Overall Process (one component)

Proxy starts up on remote machine

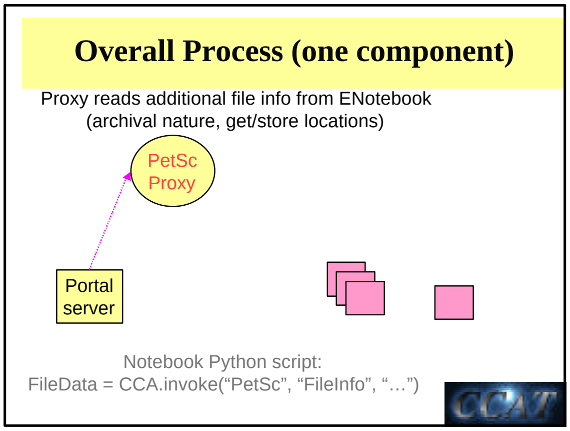
Notebook Python script:
PetSc = CCA.createInstance(PetSc)



## Overall Process (one component)

Server send application configuration data to proxy.
Proxy reads XML descriptions for app files from app developer

Notebook Python script:
CCA.execute(PetSc)



### Overall Process (one component)

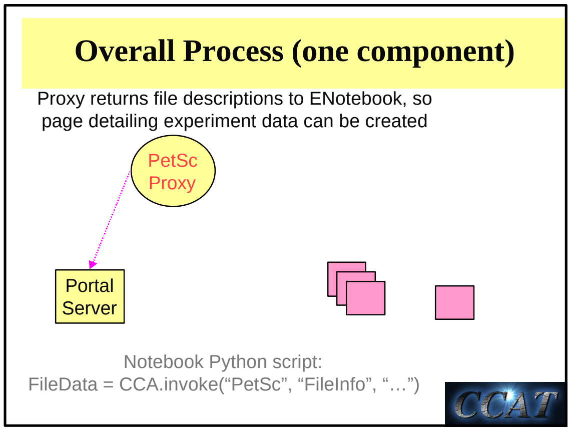
Proxy reads additional file info from ENotebook (archival nature, get/store locations)

PetSc Proxy

Portal server

Notebook Python script:
FileData = CCA.invoke(“PetSc”, “FileInfo”, “…”)

CCAT



### Overall Process (one component)

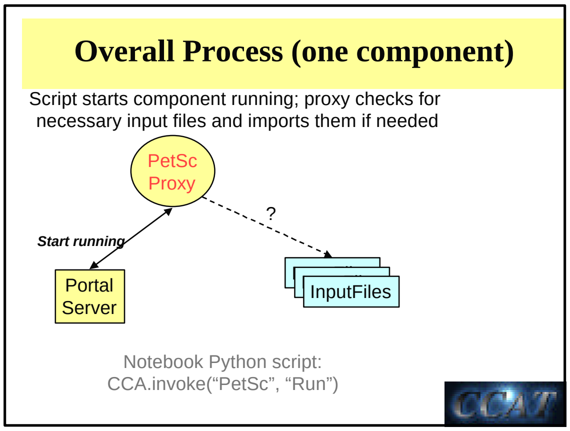
Proxy returns file descriptions to ENotebook, so page detailing experiment data can be created

PetSc Proxy

Portal Server

Notebook Python script:
FileData = CCA.invoke(“PetSc”, “FileInfo”, “…”)

CCAT



### Overall Process (one component)

Script starts component running; proxy checks for necessary input files and imports them if needed

PetSc Proxy

*Start running*

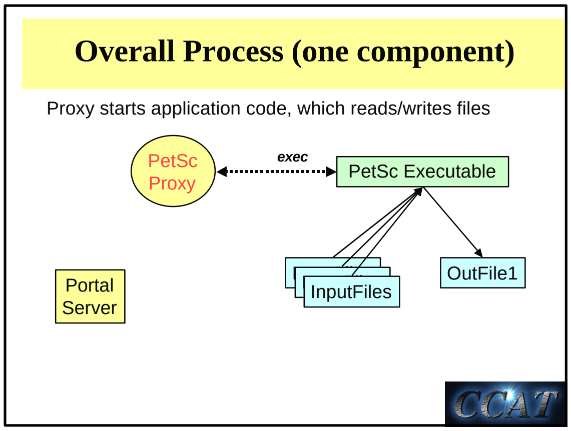
?

Portal Server

InputFiles

Notebook Python script:
CCA.invoke(“PetSc”, “Run”)

CCAT



### Overall Process (one component)

Proxy starts application code, which reads/writes files

PetSc Proxy

*exec*

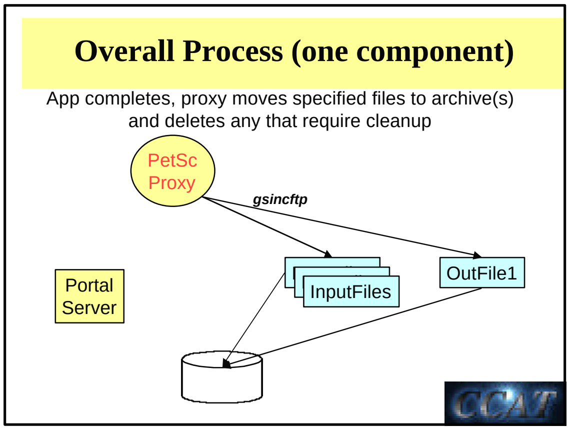
PetSc Executable

Portal Server

InputFiles

OutFile1

CCAT



### Overall Process (one component)

App completes, proxy moves specified files to archive(s) and deletes any that require cleanup

PetSc Proxy

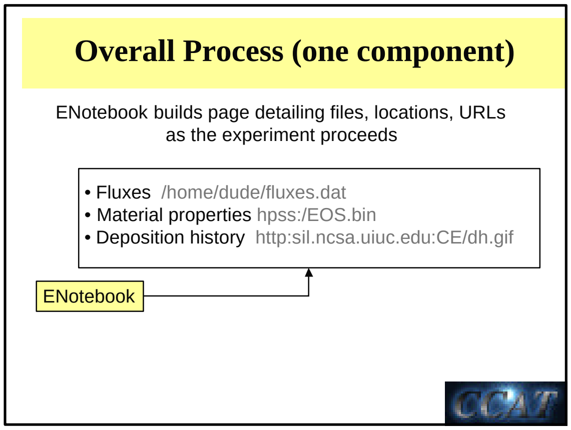
*gsincftp*

Portal Server

InputFiles

OutFile1

CCAT



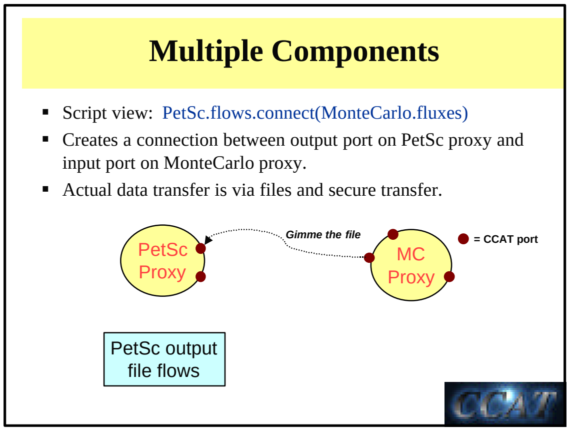
### Overall Process (one component)

ENotebook builds page detailing files, locations, URLs as the experiment proceeds

- Fluxes /home/dude/fluxes.dat
- Material properties hpss:/EOS.bin
- Deposition history http:sil.ncsa.uiuc.edu:CE/dh.gif

ENotebook

CCAT



## Multiple Components

- Script view: PetSc.flows.connect(MonteCarlo.fluxes)
- Creates a connection between output port on PetSc proxy and input port on MonteCarlo proxy.
- Actual data transfer is via files and secure transfer.

PetSc Proxy

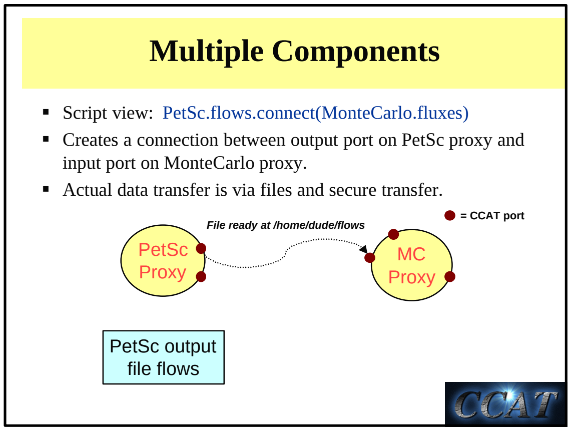
Gimme the file

MC Proxy

● = CCAT port

PetSc output file flows

CCAT



## Multiple Components

- Script view: PetSc.flows.connect(MonteCarlo.fluxes)
- Creates a connection between output port on PetSc proxy and input port on MonteCarlo proxy.
- Actual data transfer is via files and secure transfer.

File ready at /home/dude/flows

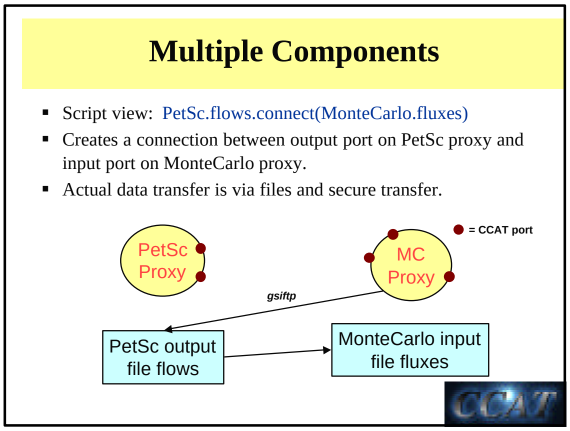
● = CCAT port

PetSc Proxy

MC Proxy

PetSc output file flows

CCAT



## Multiple Components

- Script view: PetSc.flows.connect(MonteCarlo.fluxes)
- Creates a connection between output port on PetSc proxy and input port on MonteCarlo proxy.
- Actual data transfer is via files and secure transfer.

● = CCAT port

PetSc Proxy

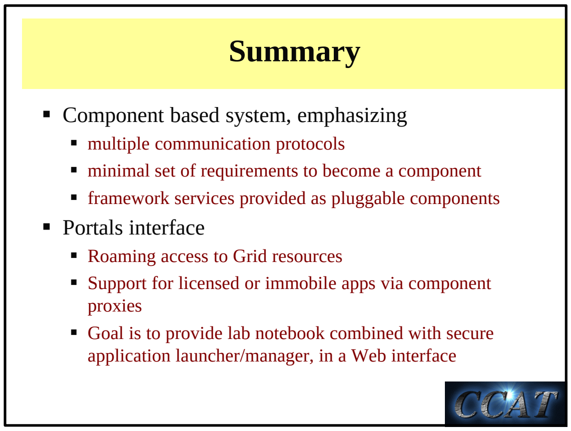
MC Proxy

gsiftp

PetSc output file flows

MonteCarlo input file fluxes

CCAT



## Summary

- Component based system, emphasizing
  - multiple communication protocols
  - minimal set of requirements to become a component
  - framework services provided as pluggable components
- Portals interface
  - Roaming access to Grid resources
  - Support for licensed or immobile apps via component proxies
  - Goal is to provide lab notebook combined with secure application launcher/manager, in a Web interface

CCAT