

# Applications Development Outside the AG

## Visualization Examples



Retreat 2003

# Outline

- Technologies used for prototypes
- Examples
  - Integration of AG and Grid resources
    - Chromium display technologies
    - Grid-aware visualization applications
  - Steps towards Grid-enabled AG Visualization applications



Retreat 2003

# Technologies

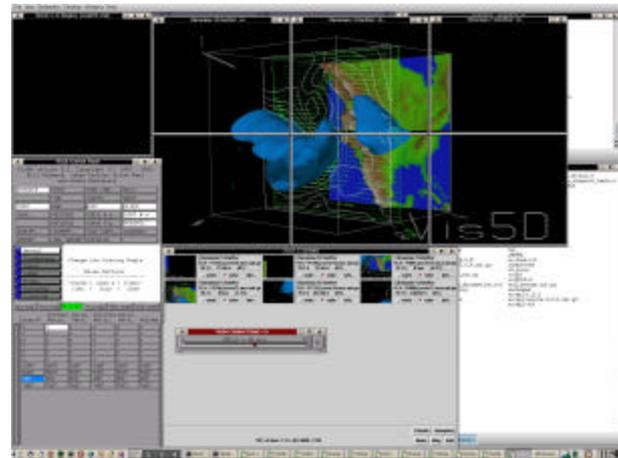
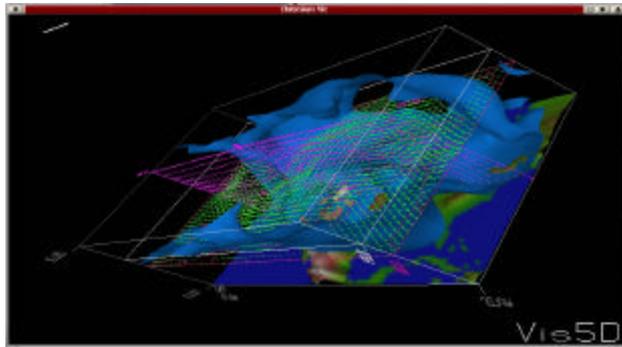
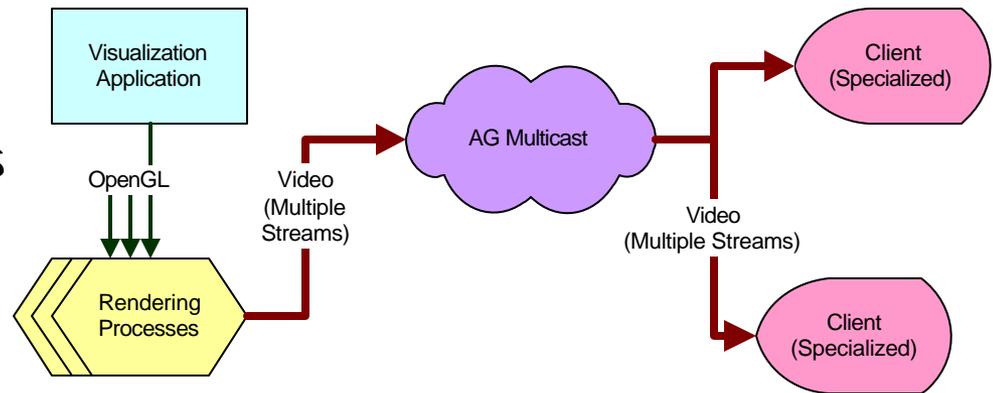
- Development
  - Python
  - Chromium
  - MPI (MPICH and MPICHG2)
  - OpenGL
  - VTK
  - Globus
- Clients
  - AG Venue Client
  - OpenMASH Vic (h.261)
- Communication
  - Multicast
  - XMLRPC
  - SOAP / WSDL



Retreat 2003

# Extensions of Chromium to AG

- Application: Provide high-resolution output for existing applications over the Access Grid, with a single user interacting with the application.



# Advantages

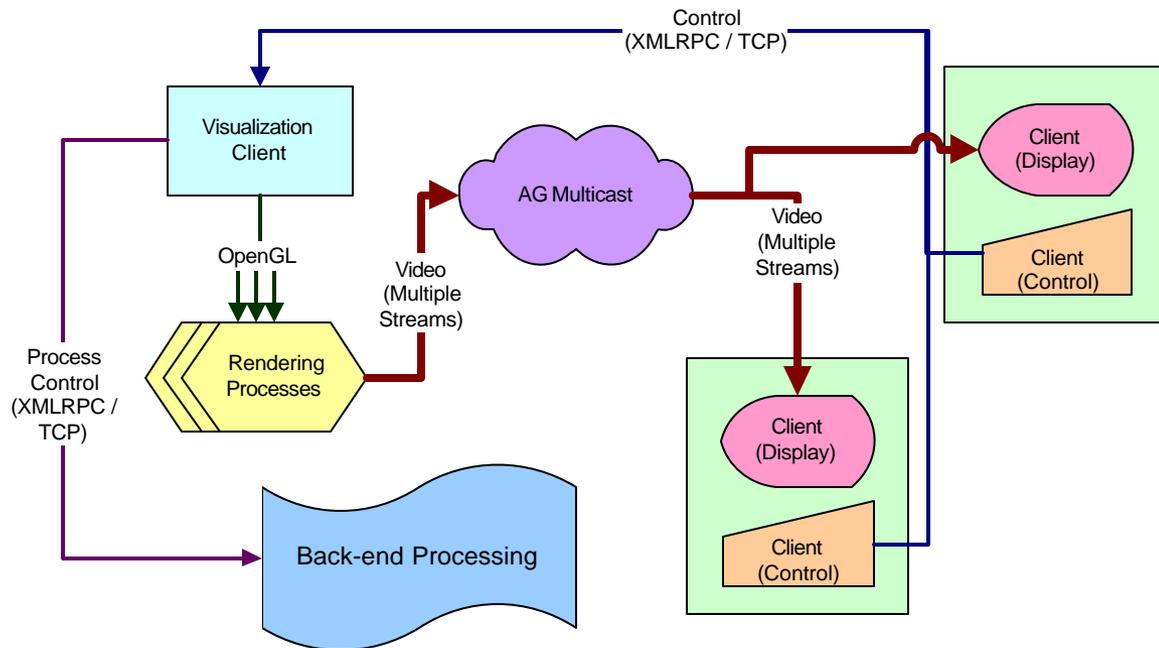
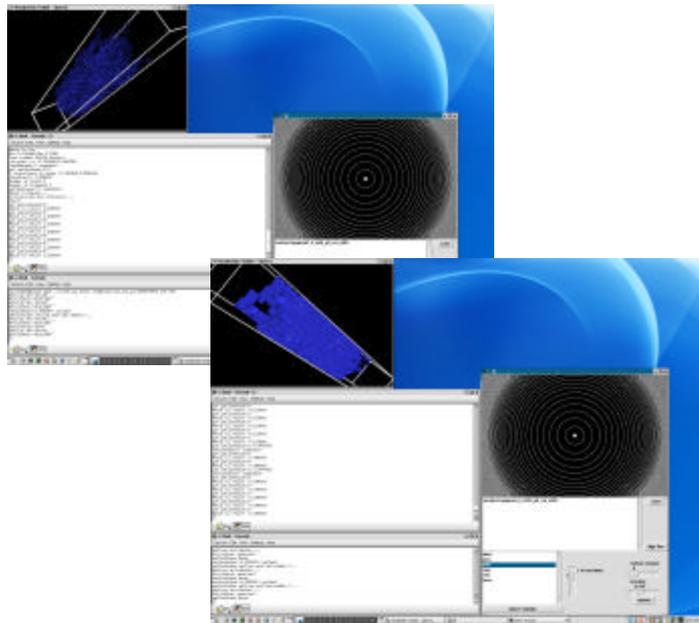
- Can be used with existing applications unmodified, even closed-source applications, provided OpenGL is used for display
- Distributed graphics via VIC provide uniform output to all participants
- Graphical output is sent using standard Access Grid streaming video mechanisms, allowing for recording via Voyager, etc.
- High-resolution output (essentially arbitrary, limited by computing resources) using multiple tiled streams
- Cluster used for rendering



Retreat 2003

# AG and Grid Application Integration

- Application: Provide high-resolution, distributed, interactive access to a reduced dataset for exploration and selection of parameters used to submit full dataset resolution visualization job requests to an automated back-end system.



# Advantages

- Reduced dataset in distributed front-end allows for manipulation of very large datasets, in a distributed setting, without losing interactivity
- Customized client allows for wide range of dataset interactions
- Many hybrid compute models possible for full-dataset 'end product' visualization
- Many users can simultaneously interact with the data to produce a desirable view before full-dataset jobs are submitted for processing
- Makes use of Grid Technologies, allowing for easier integration with AGTk 2.x



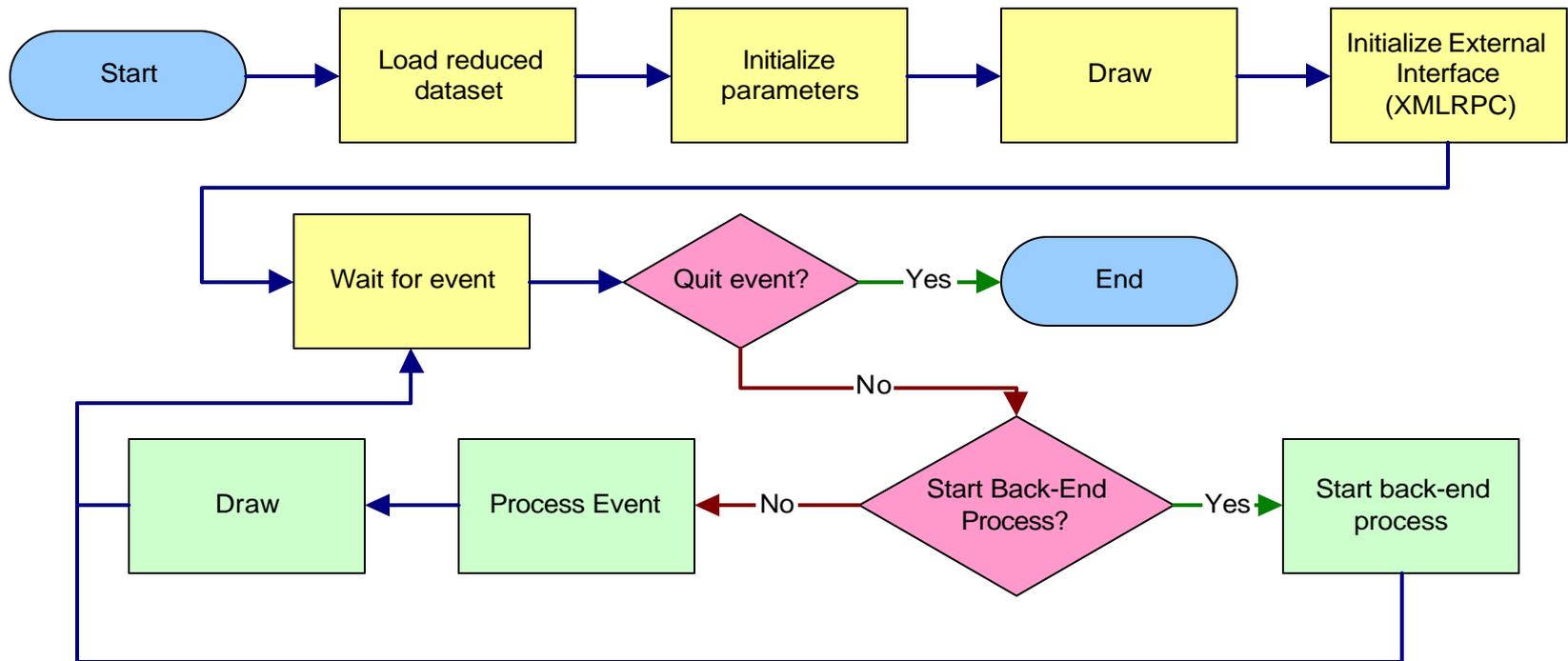
Retreat 2003

# Disadvantages

- Specialized client required for interaction
- Large, complex system with many dependencies
- Requires custom coded applications and clients

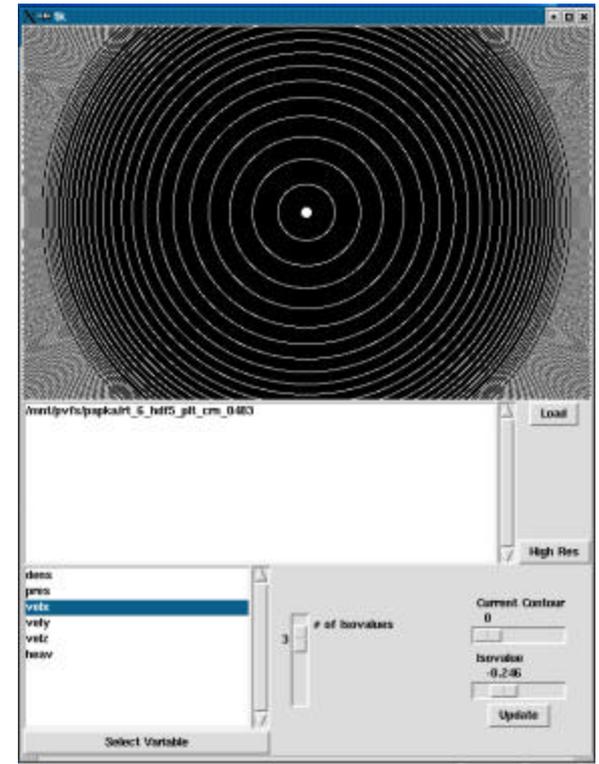


# Reduced-data Visualization Application (Specialized VTK Interactor)

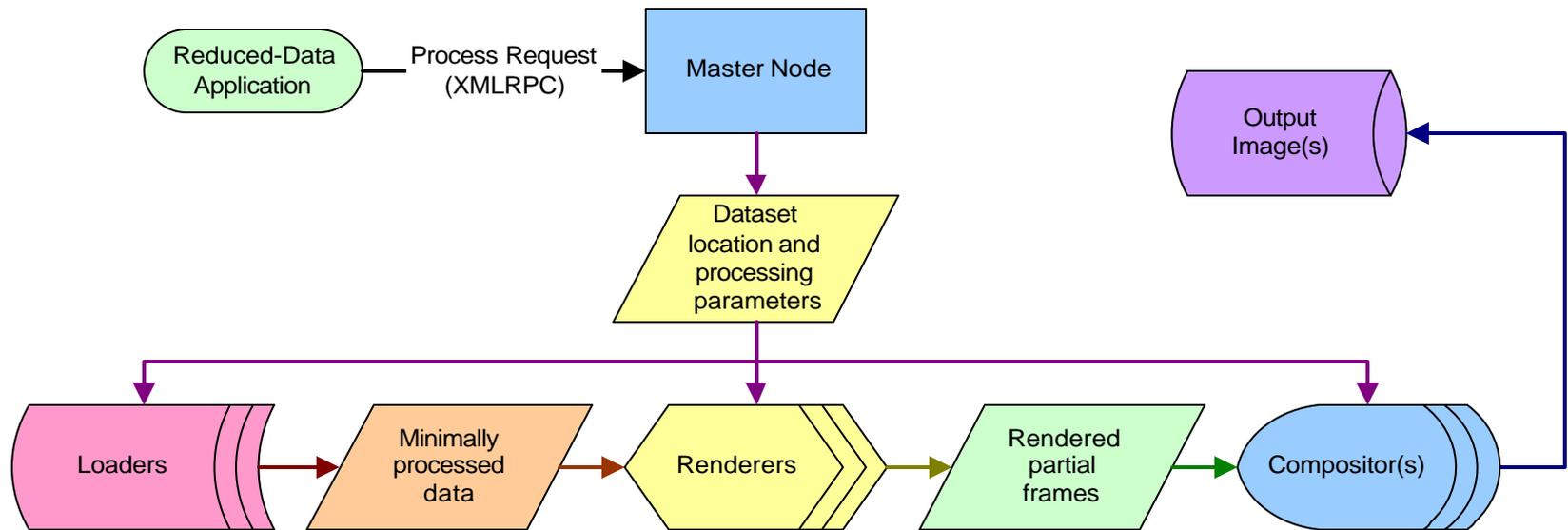


# Visualization Interaction Client

- Uses Python to produce a rich client UI
- Uses XMLRPC to communicate with the Distributed reduced-data visualization application
- Regularly updates itself from visualization app state, allowing for many parties to simultaneously interact



# Back-end Processing Engine



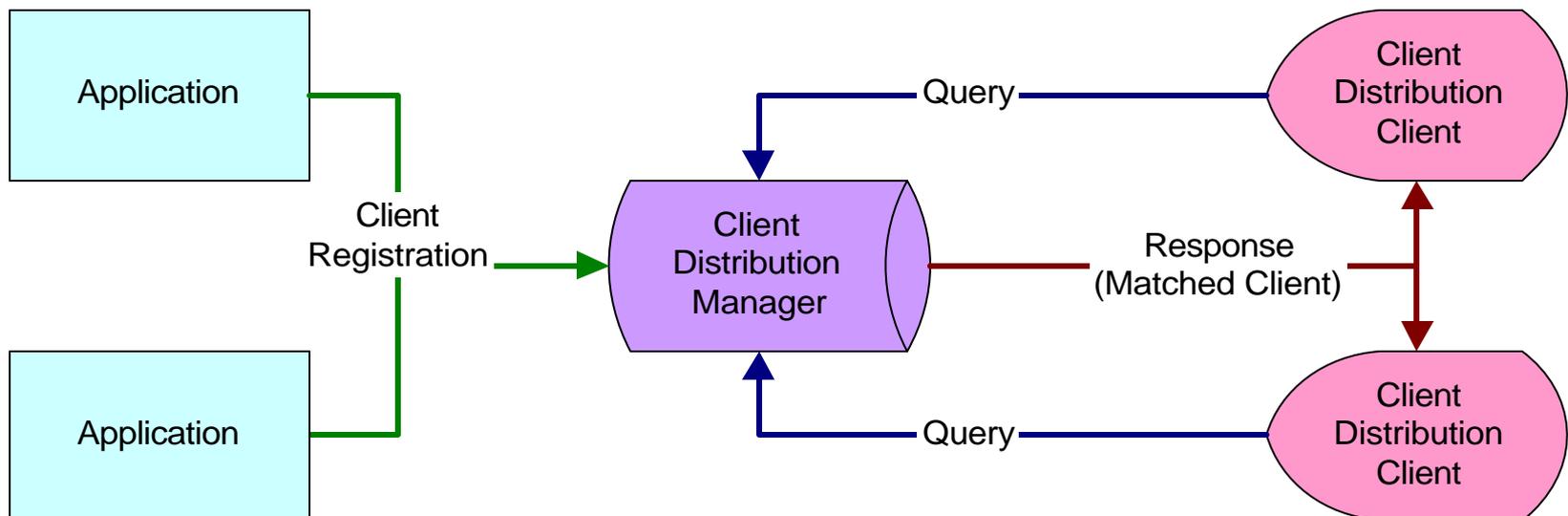
# AG Integration

- Grid-enabled nature of AGTk 2.x allows for remote invocation of various components
- Personal credentials, used for AG authentication/security, also used for job management in Globus environment
- (Prototype) Verified above capabilities by spawning a complete job (all three components) from a single, simplified user interface



# Steps Towards Embedded AG Application

- Application: Provide a customized client for on-demand, with minimal dependencies.



# Advantages

- User in an Access Grid session need not have client prior to use
- Common set of minimal dependencies can provide support for a wide range of clients
- Client version skew is minimized by providing a fresh, most-recent copy of the client each time it is used



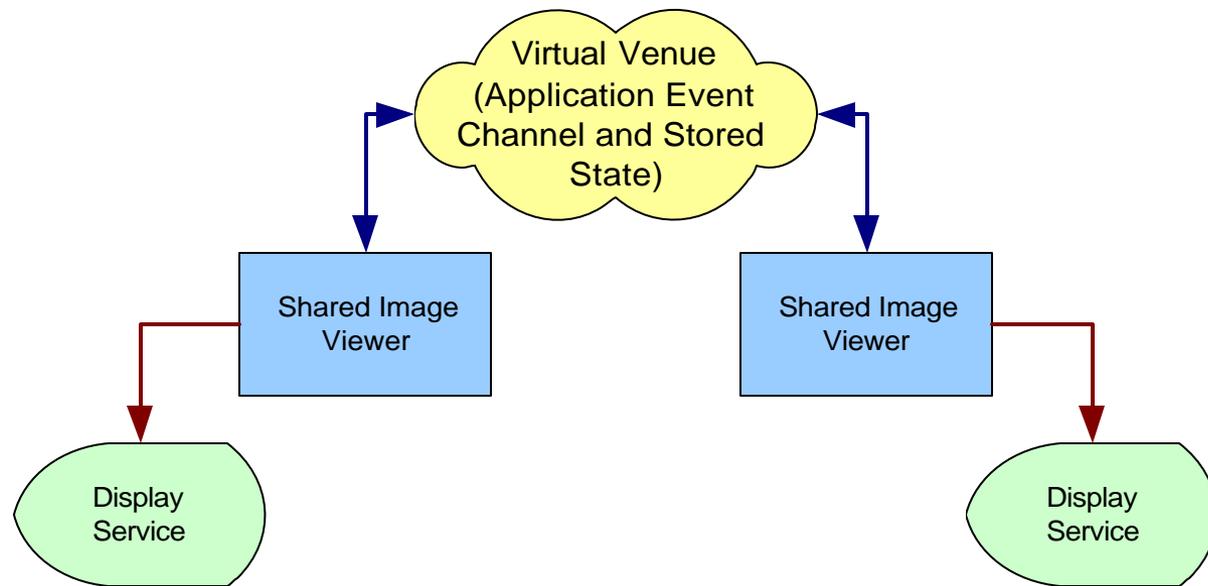
# Disadvantages

- Requires pre-installed minimal client distribution component
- Requires common dependencies to be pre-installed (such as Python, Tkinter, etc.)
- Requires custom coded applications and clients



# Shared Image Viewer

- Application: Provide distributed, shared image viewing capabilities utilizing the AGTk 2.0 Infrastructure.



# Advantages

- Participants in a collaboration can view arbitrary image data with venue-based synchronized updates
- Image display can be optimized for different physical display systems
- Proper local application is used to render the images based on the platform of each user



Retreat 2003

# Application Model

- Join the Shared Image Application Venue component
- Pick a Display Service to display to (if multiple Display Services are available, offer choice to the user)
- Get any currently displayed image from the saved state
- Wait for events requiring an update of the viewed image
  - If locally requested, broadcast the event and update the shared state



# Application Implementation

- Implemented as a set of Python classes
- Makes extensive use of utility functions and common dependencies provided by the Access Grid Toolkit
  - PyGlobus wrappers
  - SOAP wrappers
  - AG Service abstractions
  - wxPython used for display
- Entire source is 293 lines, including comments and white-space
- Written with minimal knowledge of Access Grid Toolkit internals, and no knowledge of SOAP / WSDL
- Fully secure, Grid-enabled, and integrated with Access Grid Toolkit 2.x



Retreat 2003

# Shared Image Viewer

## Shared App (Venue Communications)

```
class SharedImageViewer( wxApp ):
    def __init__( self, venueUrl, profile,
                 nodeServiceURL="https://localhost:11000/NodeService" ):
... code deleted ...
    self.venueProxy = Client.Handle(venueUrl).GetProxy()
    self.nodeServiceProxy = Client.Handle(nodeServiceURL).GetProxy();
    self.privateId = self.venueProxy.Join(profile)
    # Retrieve the channel id
    (self.channelId, eventServiceLocation) =
    self.venueProxy.GetDataChannel(self.privateId)
    # Retrieve list of node services
    serviceList=self.nodeServiceProxy.GetServices()
    # Parse service list, remove all non-display services
    self.displayServices=[]
    for service in serviceList:
        for capability in service.capabilities:
            if capability.role=="consumer" and capability.type=="display":
                service.capabilities=Client.Handle(service.uri).GetProxy().GetCapabilities();
                self.displayServices.append(service);
    # Subscribe to the event channel
    self.eventClient = EventClient.EventClient(eventServiceLocation,
        self.channelId)
    self.eventClient.start()
    self.eventClient.Send(Events.ConnectEvent(self.channelId))
    # Register the 'view' event callback
    # The callback function is invoked with one argument, the data from the
    call.
    self.eventClient.RegisterCallback("view", self.ViewCallback )
    # Create View status panel
```

```
... code deleted ...
    self.viewer=Viewer(self.frame,-
        1,self.targetSurface,self.displaySize);
... code deleted ...
    # Browse to the current url, if exists
    currentImage = self.venueProxy.GetData(self.privateId,
        "image")
    if len(currentImage) > 0:
        self.viewer.display(currentImage)
... code deleted ...

def newImage(self,data):
    # Send the event
    self.eventClient.Send(Events.Event("view",
        self.channelId, ( self.profile.publicId, data ) ))
    # Store the URL in the app object in the venue
    self.venueProxy.SetData(self.privateId, "image", data)

def ViewCallback(self, data):
    # Determine if the sender of the event is this
    component or not.
    (senderId, image) = data
    if senderId == self.profile.publicId:
        print "Ignoring %s from myself" % (image)
    else:
        print "Displaying ", image
        self.viewer.display(image)
```



Retreat 2003

# Shared Image Viewer

## Viewer (locally displays the image)

```
class Viewer(wxPanel):
    def __init__(self, parent, id, displayContact, displaySize, frame = None):
... code deleted ...
    def display(self, viewURL):
... code deleted ...
        try:
            if viewURL.startswith("https"):
                DataStore.GSIHTTPDownloadFile(viewURL, tfilepath, None, None)
            else:
                my_identity = GetDefaultIdentityDN()
                DataStore.HTTPDownloadFile(my_identity, viewURL, tfilepath, None, None)
        except DataStore.DownloadFailed, e:
            wxCallAfter(wxLogError, "Got exception on download")
        if self.displayContact.startswith("Win32"):
            os.system("start %s"%(tfilepath))
        else:
            import posix
            if self.pid_set:
                posix.kill(self.pid_set,9)
                self.pid_set=0
            self.pid_set=posix.fork()
            if self.pid_set:
                return
            else:
                posix.system("display -display %s %s &%"("localhost:" + self.displayContact[6:].split(":")[-1], tfilepath))
                posix._exit(0)
```



Retreat 2003

# Future

- More AG functionality
  - More applications like the Shared Image Viewer
  - Integration of complex visualization applications with the AG
  - Services implemented using Grid standards, scoped within Virtual Venues



Retreat 2003

# Questions?



Retreat 2003

# Contact Information

Questions or Comments regarding any of the content in this presentation may be directed to:

Justin Binns

Futures Laboratory / Argonne National Laboratory

[binns@mcs.anl.gov](mailto:binns@mcs.anl.gov)



Retreat 2003