### Photo Sharing Application

We would like to share photos with friends and family. The solution we will create is a Photo Sharing analytical service. This service allows us to create photo galleries and securely share photos with others.

Our model contains semantic information so that model users know exactly what each attribute means. We clarify the meaning of each attribute by assigning CDEs from the caBIG caDSR. We point out that we have not registered our model in the caDSR as a complete project, but that is normally done (we haven’t done this just because our model is for training purposes only). In order to create an analytical service with semantic annotations (e.g., semantic information on objects passed over the Grid), we need a model registered in caDSR. So, the service we build today will not have semantic annotations.

Next section of slides needs to give complete details on our solution for authentication and authorization, along with insight on how authorization works in the service itself. Optional talking point (backup slides) on using PDPs for authorization instead of mixing authorization in with our business logic.

Photo Sharing Features:

* Ability to organize photos into galleries
  + Can create new galleries
  + Can upload photos to chosen galleries
  + Can run simple image processing operations on selected photos (zoom)
  + The gallery owner (creator) can specify exactly who can add and view photos in each gallery
  + The gallery owner can override gallery view permissions for specific photos by specifying which users can retrieve images
* The service caches photos that are uploaded to galleries, but does not persist data
* Gallery owner has complete control over who can see exactly what photos
  + We will configure our Photo Sharing service so that anyone can create galleries to share with others. But, we want to know who is creating galleries, so we want to make sure everyone who can create galleries has logged on (require authentication)
  + Specify who is allowed to add photos to photo galleries. We will configure our Photo Sharing service so that by default only the owner (creator) of the gallery can add and view photos. The gallery owner must grant view and add privileges for others to see photos.
  + Only the gallery owner can modify permissions for a gallery (add photos, view photos, set image permissions).
  + Data-level authorization: The gallery owner can specify exactly who is allowed to retrieve a specific image (overriding gallery view permissions).
    - Example scenario: I have two photo galleries that I want to share. The first gallery, “Business retreat”, includes photos from a recent vacation. I’m happy to share these photos with friends and family (everyone). The second gallery, “Vacation” is of photos from a recent vacation. I want to make sure that only friends can see this gallery, and for some pictures, I want to share them only with friends who are present in each picture.

Technical Summary of the Grid Service we will build

* Our service uses fine-grained authorization
  + Service-level authorization: require that anyone contacting the service has logged on to the Grid (require authentication and connection as non-anonymous user). This is the security setting both for the main service and for the Gallery context.
  + Operation-level authorization: Only the gallery owner can modify privileges (this applies to the following operations: grantViewGalleryPrivileges(User), revokeViewGalleryPrivileges(User), grantAddImagePrivileges(User), revokeAddImagePrivileges(User), grantImageRetrievalPrivileges(ImageDescription, User), revokeImageRetrievalPrivileges(ImageDescription, User)).
  + Context-specific operation-level authorization: ImageDescription[] listPhotos() also is allowed only by gallery viewers. Furthermore, if an image isn’t allowed to be seen by the requestor, then the ImageDescription[] is partially complete (filtered for the user). addImage(Image) is allowed only by gallery adders.
  + Data-level authorization: Each image has permissions that override the gallery viewing permissions.
    - By default, all gallery viewers can retrieve all images. However, in the event that an image has privileges set for it, then those privileges override the gallery defaults. Example: Tom is in the gallery viewers group (has been granted view privileges). The gallery owner, Bob, wants to make sure only Sue and Bob can retrieve a specific image X. When Tom calls listImage, image X is filtered from the list (Tom doesn’t know it’s in the gallery). Moreover, if Tom tries to retrieve that specific Image, he gets an AuthorizationException, as he is not in the group for that specific Image (only Sue and Bob are in that group).

Permissions to create stems and groups underneath…

We have a stem for this tutorial: “Training:photoSharingcaBIGAnnualMeeting2009”. Under this stem, each service has its own stem to manage permissions. This stem will hold all galleries that the service creates. When a service client creates a gallery, the service creates a new sub-stem underneath the service stem.

Work In Progress: However, the ability to create stems is given to specific users by a stem admin. What we do is create a stem administrator user (a new Dorian user specifically for photo sharing tutorial privileges for the 2009 annual meeting named “photoadmin”) and give them permissions to manage the stem. Then we delegate their credentials to a photo sharing admin group (“Training:photoSharingcaBIGAnnualMeeting2009:photoSharingTutorialAdministrators”  
. Then each tutorial attendee joins that group (can self-join). To give this ability, we set the “GrouperAll” identity on the group to have the following privileges: “Read”, “View”, “Optin”, “Optout”. Then each tutorial attendee retrieves the photo sharing stem admin credentials from CDS. Acting as the photo sharing stem admin, they then create their own personal stem for sharing their photo galleries. E.g., “jpermar photo sharing” is a stem underneath “Photo Sharing Tutorial caBIG Annual Meeting 2009”.

## Service Creation

Install caGrid configured for training, register user, setup secure container.

Note: for Introduce service creation, use screenshots Justin created (in zip). With every demo step (create user, add photo, etc.), go to grid grouper and take a look at what the service did.

TODO: add GUI person (Dave)

TODO: make a phase 2 that has data-level permissions

TODO: tutorial note… the tutorial taker should use GAARDS UI to add their friend’s identity to the view and add image groups tht are in grid grouper. (yes can do this via our GUI/service API, but this shows them they can interact with grid grouper).

Fix bugs: Axis serialization, GridGrouper() constructor, and service identity retrieval feature.

TODO: add service identity property

TODO: create GalleryRegistration service where they send host identity and grid identity. Makes a new stem and adds both user and service as STEM administrators. Tutotrial taker has an ant task to call this service and send their 1) hostname, 2) host identity and their 3) user identity… add both identities as admin on a new stem that is named hostname

Service creation:

1. Copy mapping extension to introduce and install it.
2. Go through steps as outlined in screenshots
   1. Create service
   2. Add gallery context with security
   3. Set main context security
   4. Add createGallery operation to main service
   5. Add listGalleries operation to main service
   6. Add getGalleryName operation to Gallery context
   7. Add addImage operation to Gallery context
   8. Add listImages operation to Gallery context
   9. Add getImage operation to Gallery context
   10. Add grantViewGalleryPrivileges operation to Gallery context
   11. Add revokeViewGalleryPrivileges operation to Gallery context
   12. Add grantAddImagePrivileges operation to Gallery context
   13. Add revokeAddImagePrivileges operation to Gallery context
   14. (Phase 2) Add grantImageRetrievalPrivileges operation to Gallery context
   15. (Phase 2) Add revokeImageRetrievalPrivileges operation to Gallery context
   16. Add service properties
3. Copy photosharing.jar to service lib
4. Add required caGrid jars:
   1. Copied gridgrouper libs (common, client, stubs)
   2. Copied ./projects/gridgrouper /ext/dependencies/service/jars/grouper.jar
   3. Copied ./projects/gridgrouper/ext/dependencies/service/jars/subject-0.2.1.jar
   4. Copied caGrid-mappingExtensions-1.3.jar to service lib.
   5. ADDED ALL JARS TO ECLIPSE CLASSPATH (don’t use eclipse for demo tho… package Notepad++ in the zip)
   6. Had to fix sdk type mappings (wsdd files).
   7. Added additional jars needed for mapping extension: sdk-client-framework.jar and castor-1.0.2.jar.
5. Create new PhotoSharingGridGrouper class (in org.cagrid.demo.photosharing.service package)
6. Modify PhotoSharingImpl.java constructor and methods.
7. Modify PhotoSharingResource.java
8. Modify GalleryResource.java.
9. Modify GalleryImpl.java
10. Modify PhotoSharingClient.java
11. Copy osu medcenter logo and other images to service for demonstration.
12. Update service.properties and deploy the service. Deploy using GUI, as it formats service.properties properly.