

DPACS: A Distributed Scaleable Image Archival and Browsing Infrastructure. *

Shannon Hastings and Scott Oster

Department of Biomedical Informatics, The Ohio State University
Columbus, OH, 43210
{hastings,oster}@bmi.osu.edu

Abstract

Images are a common data type in many different aspects of science and medicine. Images are used in analysis from oil reservoir production quantification to lung cancer diagnosis. Information can be abstracted from images in many ways using a wide variety of analysis techniques. As non-invasive data collection technology grows in many different areas we begin to see more and more image acquisition techniques yielding more and more image data at higher resolutions. Managing this data efficiently is not only a large and complex problem but a problem where, if solved efficiently, yields critical new findings. Image storage, querying, and retrieval is a critical component to enable the use image data in a scientific setting. From doctors in hospitals to research scientists in their fields, efficient management of large quantities of ever growing sizes of image data help lead them to there finding faster saving them time and money while advancing there fields. In this paper we present a distributed framework for managing images in a massive scale, leveraging new advances in grid middleware coupled with large interconnected computation and storage hardware.

1 Introduction

The Distributed Image Archive System (DPACS) is designed to support generic image archival and management in a grid wide environment. The DPACS system is built using the Mobius middleware framework for data management, the java advanced imaging API (JAI), and many other common components.

*This research was supported in part by the National Science Foundation under Grants #ACI-9619020 (UC Subcontract #10152408), #EIA-0121177, #ACI-0203846, #ACI-0130437, #ANI-0330612, #ACI-9982087, Lawrence Livermore National Laboratory under Grant #B517095 (UC Subcontract #10184497), NIH NIBIB BISTI #P20EB000591, Ohio Board of Regents BRITC #BRIT02-0003.

Advances in acquisition techniques, reduction in commodity hardware prices, and the need for non-invasion data gathering techniques have lead to an enormous growth in not only the size and resolution of single images but also in the number of images. These images are only useful in their respective domains if the can be managed efficiently and effectively.

The medical field has witnessed advances in many different areas which involve imaging. The new drive to use non-invasive techniques for patient diagnosis has caused many advancements in image acquisition techniques. Computed Tomography (CT), Magnetic Resonance Imaging (MRI), Molecular Cytogenetics, Scanned Light Microscope slides, as well as many other variants of these techniques have become a common tool used in patient diagnosis. These advancements in acquisition techniques have increased image resolution which, intern, increases the average size of image studies. Also, because these techniques are becoming more and more useful the total storage requirements for a particular institution continue to grow making management of data on such a large scale more and more difficult. Although clinical images such as CT, MRI, or Light Microscope, for example, currently make up that largest percentage of most hospitals image data; there is also image data from many other facets of medicine such clinical research in proteomics and genomics.

In other scientific fields, ranging from drug discovery to satilite image analysis, advances have also increased the size and number of images. These areas, as in the medical domains, are pushed by research demands for better resolution, larger data sets to mine from, and the basic need to obtain more meaning from the data. New research on satellite image, crowd video, baggage scan, fingerprint, and digital face analysis, driven by homeland security requirements, have drastically changed the information asset of government intelligence from mostly text to now both text and imagery together to form a rich ontology of knowledge. Managing and mining this data effectively will be critical in the ability for government to detect and impede possible threats as well as maintain current and intelligent knowledge about

the current security state.

As we have stated, being able to store images of such great size and number size is critical, however, the ability to analyze the data efficiently and effectively is just as critical. This drives the requirement for being able to annotate and index the images with user defined metadata. This metadata will be used in querying , data distribution, and representing knowledge about the data. Images might be organized in volumes of 2D tiles, 3D volumes, 4D volumes dependent on time, and so on. Each image may have some particular metadata attached to it which is relavent to the image itself or maybe to the knowledge the image might represent. In the medical field, for example, the image may be organized by patient id and study id, and may have basic medical patient demographic metadata attached to it like age, sex, height, or weight to name a few. Every domain with image data may have their own metadata information to attach to the image making it a critical requirement that they system be flexible and be able to store and query these images with their metadata so that complex queries, which use the metadata and image information, can be executed. Every image data type, such as PNG or DICOM, may also have a set of metadata which can be attached to the image at that level, separate from the user-defined metadata.

To complicate this scenario now we add the requirement that the system be able to scale up to grid levels. Image storage services across the world with terabytes of images and metadata distributed throughout them must be able to share data, query each other, and ensure a quality of service (QoS) metric, as well as maintain security constraints. With this requirement, the system becomes drastically more complicated but also much more powerful. Scientific data integration across multiple disciplines is becoming more and more prevalent in scientific research areas. With the new growing uses of imagery as stated above, being able to integrate this imagery across these domains enables new ways to analyze the data increasing the usefulness of the data.

2 Distributed PACS

The DPACS system consists of a front end client application, and a collection of federated back end Mobius Mako servers. The Mako servers provide the image and metadata storage and retrieval. The client application be be used by user to upload, query, and inspect the archived data.

2.1 DPACS Client Application

The DPACS client front end, shown in Figure 1, provides an intuitive, centralized view of the distributed PACS back end. It attempts to create a local workspace for the user by hiding the details of retrieving and locating datasets within the back end.

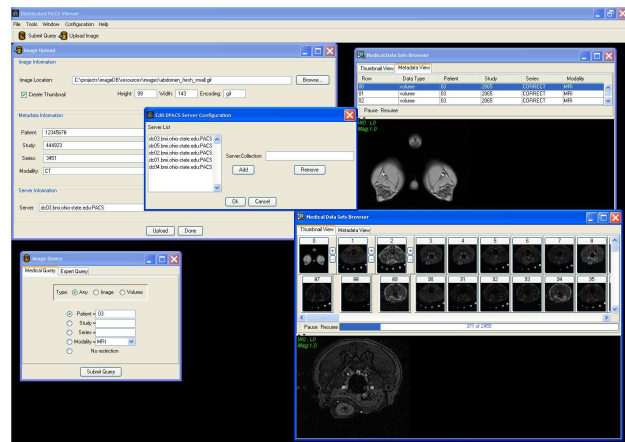


Figure 1. DPACS Client Screen Shot

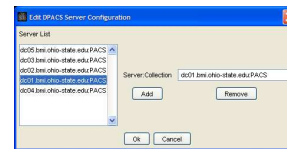


Figure 2. Server Configuration Screen Shot

2.1.1 Application Component Description and Usage

In order to use the DPACS client application, it must be configured to know about the available back end servers. This task is accomplished by using the simple server configuration tool, shown in Figure 2. The user enters combinations of Mako server and a corresponding XML Collection name. The list of configured tuples controls the servers that will be contacted for dataset queries, and those which will be available for the user to upload data to.

The user may create new datasets via the DPACS client's image upload capability. This tool, shown in Figure 3, provides a simple mechanism whereby local images can be associated with medical metadata and uploaded to the back end. This is a quick and easy to use tool, which is useful for creating small datasets or adding a small number of images to an existing dataset; it is not convenient for submission of large datasets. For such tasks, a command line tool is provided to facilitate batch submission of large datasets spread across the back end.

Once a number of datasets have been uploaded to the back end, a critical task is locating the desired datasets and re-aggregating a given dataset when it needs to be inspected. To facilitate this task, a simple query tool, shown in Figure 4, is provided. It gives the user the ability to query for datasets by type, and by particular values of metadata. Additionally, for those users familiar with XPath, an "expert"

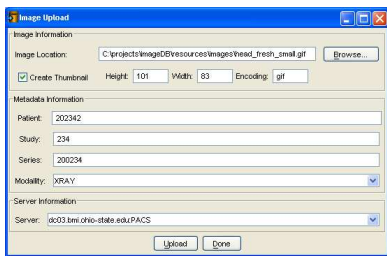


Figure 3. Image Upload Screen Shot

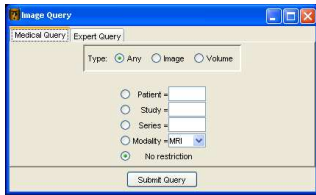


Figure 4. Dataset Query Screen Shot

mode is provided for querying by complex XPath expressions. Once the user has configured the desired query, it can be submitted to the backend and the results will be shown in a new dataset browser. Here the power of the Mobius backend is shown, as queries are broadcast to the backend nodes, and soft references to the datasets are quickly returned.

The results of queries are displayed in a medical dataset browser, shown in Figure 5, which provides mechanisms for traversing both volumetric and single image datasets as via thumbnails and a metadata viewer. The browser provides two synchronized views of the collection of datasets. The first view is image thumbnail view which shows a small representation of each image in a returned dataset, and a candidate image from each returned volumetric dataset. This provides a compact visualization when numerous volumes are returned. The user is also able to quickly scroll through the thumbnails of a given volume without needing to retrieve the complete image. The unobtrusive controls for this can be seen in *dataset 76* in Figure 5. The second view is a metadata view, wherein the available metadata for each dataset is displayed in tabular form. When the user selects a given dataset in either view, it is selected in both views, and the current image is displayed from that dataset. For images, this is just the full image data of the thumbnail that was selected, but for volumetric datasets, this is full image data of the currently selected volume thumbnail. The image data is displayed below these views in an image inspection window. The individual images can be manipulated using select operations from the Java Advanced Imaging library. The user can conveniently adjust the window/level of the image by dragging the left mouse button across the image's

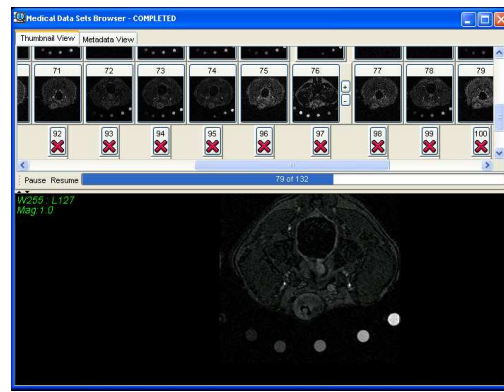


Figure 5. Medical Datasets Browser Screen Shot

surface. Changes in the X plane are mapped to alterations of the image's level, whereas changes in the Y plane are mapped to alterations of the image's window. Dragging the left mouse button while depressing the control key on the keyboard the user is able to quickly manipulate the image's scale. Additionally, the user can quickly change the placement of the image by using the right mouse button. By utilizing the scale and placement operations in conjunction, the user is able to quickly inspect a detailed feature.

2.1.2 Application Component Optimization

Displaying and navigating a large number of image datasets in a relatively small amount of screen area is a difficult problem. Users of the DPACS application will routinely need to have several dataset browsers open, and may prefer to leave several query and upload windows open for easy access. Considering that one query may result in several thousand images, it becomes important to optimize the available screen area. The first approach DPACS takes to deal with these issues is the use of the familiar and convenient Multiple Document Interface (MDI) paradigm. MDI allows a single application to work with several "documents" at a time, while providing mechanisms to organize the group of documents. In the case of DPACS, each tool and dataset result, is treated as a document. This allows several instances of the same tool to be running at the same time, and give the user a convenient interface for manipulating his or her whole workspace, while maintaining the ability to have anyone one tool demand the entire workspace. Furthermore, many of the DPACS client tools provide detachable tool bars which allow the user to drag a portion of the tool into its own frame to make more room for other portions of a tool (such as image display). Additionally, the panels that are not detachable are collapsible so that they

can be completely hidden from view and then expanded when needed. Lastly, whenever possible, mouse gestures and keyboard commands are used to manipulate the tools instead of space consuming graphical components.

Accessing and displaying large datasets over a network of distributed servers can quickly become problematic for clients. Issues such as network delays, large data sizes and quantities, and multiple servers can make it difficult for a client application to sustain a pleasant and consistent working experience for the user, as response times can easily become unreasonable. The DPACS client application utilizes several techniques to overcome these adversities. The first important feature the application provides is the extensive use of background threading for tasks which communicate with network resources. One of the most frustrating aspects of using applications which communicate with network resources is when they are unresponsive because of an unexpected network delay or timeout. The DPACS application utilizes an active thread pool which allows sequences of tasks to be immediately performed in the background, while the main application thread continues to process user inputs and refresh the display. Using the thread pool allows several requests to take place concurrently and still give the user the power to control the application. It also ensures that limited resources are used, as the threads are immediately returned to the pool when they are finished, so they can be reused. Another powerful feature is the use of "on demand", lazy data loading. All data within datasets is left on the back end until it is absolutely needed. Only a reference to the data is stored in the client. The dataset object model abstracts this concept by presenting get methods which cache and immediately return data if it is present, and transparently load the needed data from the back end when it is not. This allows the application components to work with the datasets as though they are all locally present, and the object model handles the requests and local storage when they are not. The dataset browser takes full advantage of this feature, and is able to present a large amount of data, very quickly. Once it receives the references to the resulting datasets from a query, it displays placeholders, seen in Figure 5 as red Xs, for each. The user is able to click on any of the placeholders, and its thumbnail, metadata, and current image are immediately loaded and displayed in the image inspection panel. Concurrently, a background thread begins working on retrieving the thumbnails and metadata of each of the datasets. The progress of this task can be seen by the progress bars below the previews. Since there may be a large amount of previews to retrieve, and the user may want to use all of its computer's processing power to inspect an image before all of the previews are retrieved, controls for pausing and resuming this background job are presented. At anytime the user may press the pause button to yield this process, and then resume it when the computa-

tionally intense task they wish to execute is complete. Additionally, the client executes prefetching of image thumbnails when a user is navigating a volumetric dataset. In this technique, a background thread retrieves the next several thumbnails in the volume sequence. This allows it to appear as though the data exists locally, while not having to retrieve excessive amounts of unneeded data. The combination of all these features give the user fine-grained control over the computational resources and screen area of the application. By hiding the complexity of communicating with a federated back end, and handling the problems which can occur, the DPACS client application presents a user-friendly environment for working with a very large collection of image datasets.

2.2 Distributed Back end Storage System

The distributed back end storage system of DPACS relies heavily on the Mobius middleware. The Mobius middleware system is a distributed service architecture enabling data integration and sharing of data in a grid environment. Mobius provides the service API and protocol for storing, versioning, and linking user defined data models called the Global Model Exchange (GME). Using those models Mobius provides a service API and protocol for creating a set of federated data storage services, which store and retrieve data instances, called the Mako. The Mobius MakoDB service adheres to the Mako API and dynamically creates data storage services built from the user defined data models.

DPACS uses the Mako/MakoDB framework of Mobius in order to create the distributed image storage service infrastructure. There is one main schema that DPACS uses to define some basic image types used for storing image in the system. In this demo we have added a schema which uses the core image types schema and extends them to create data models which are relevant to the storage of images in the medical domain.

2.2.1 Mako

Mako is a service that exposes data resources as XML data services through a set of well-defined interfaces based on the Mako protocol. A data resource can be a relational database, an XML database, a file system, or any other data source. Data resources are exposed through a set of well-defined interfaces, thus exposing specific data resource operations as XML operations. For example, once exposed, a relational database would be queried through Mako using XQuery as apposed to querying it directly with SQL. Mako provides a standard way of interacting with data resources, thus making it easy for applications to interact with heterogeneous data resources.

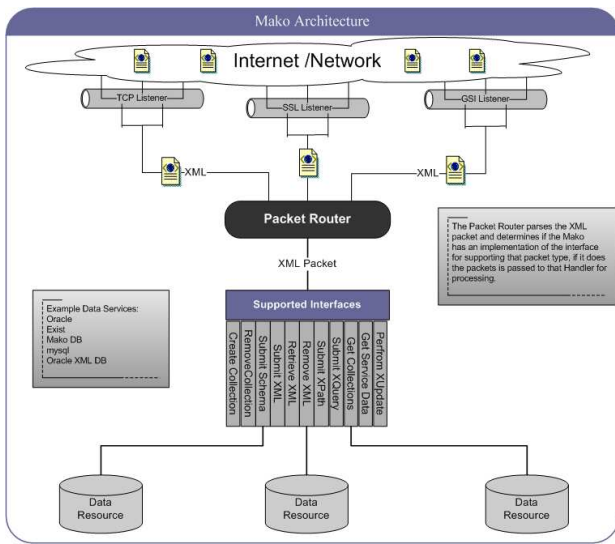


Figure 6. Mako Architecture

2.2.2 Mako Architecture

Clients interact with Mako over a network; the Mako architecture illustrated in Figure 6 contains a set of listeners, each using an implementation of the Mako communication interface. The Mako communication interface allows clients to communicate with a Mako using any communication protocol. Each Mako can be configured with a set of listeners, where each listener communicates using a specified communication interface.

When a listener receives a packet, the packet is materialized and passed to a packet router. The Packet Router determines the type of packet and decides if it has a Handler for processing a packet of that type. If such a handler exists, the packet is passed onto the handler which processes the packet and sends a response to the client.

Collection Management

In the context of XML databases, the common nomenclature for referring to a group of related instance documents is a Collection. This is very similar to the relational database concept of a table. The concept diverges slightly, in that collections can have sub collections, and collections may not have a single schema (or any at all) associated with them. Mako has three request packets for managing collections of XML documents. The CreateCollectionRequest packet is used to create a new XML collection on a Mako. Its response packet, CreateCollectionResponse is an acknowledgment of whether or not the collection could be created. The RemoveCollectionRequest packet can be used to remove a collection from a Mako. Like the CreateCollectionResponse, the RemoveCollectionResponse acknowledges whether or not the request was successful.

Finally, the CollectionListRequest packet requests a list of collections or subcollections stored in a Mako. The CollectionListResponse packet contains a list of the existing collections on the Mako or in the sub collection.

Schema Management

Each collection in Mako can be restricted to accepting XML documents from a set of certain types. This is accomplished by specifying a set of XML schemas in which a XML document must be validated against. The Mako protocol provides a method for adding and removing schemas to and from an XML collection within a Mako. The SubmitSchemaRequest packet is used to submit schemas to a collection in a Mako, and the RemoveSchemaRequest packet removes them. The SchemaListRequest packet can be used to get the list of schemas supported by an XML collection.

Document Management

The Mako protocol defines methods for submitting, updating, removing, and retrieving XML documents. The SubmitXMLRequest packet is used to submit documents to an XML collection in a Mako. Upon submission, Mako assigns each entity a unique identifier, later we will see why this identifier is important. XML documents that reside in a Mako can be updated using XUpdate, the XUpdateRequest packet is used to accomplish this.

Mako provides two methods of removing documents. First, a list of documents can be removed by specifying their unique identifiers, this is done using the RemoveXMLRequest packet. Second, XML documents can be removed by specifying an element identifying XPath expression. The XPathRemoveRequest packet is used to remove XML documents that meet an XPath expression.

The RetrieveXMLRequest packet is used to retrieve XML documents, or a subset of XML documents, from a Mako. Documents, or subsets of XML documents, can be retrieved by specifying their unique identifier. Recall each element in and XML document is given an identifier, making any subset of a document uniquely addressable. The Mako protocol also allows the level of retrieval to be specified. For example, if you think of an XML document as a tree, then given a unique identifier, one would be able to specify how many levels of children should be included in the materialization of the document. Elements containing children that are below the height specified would not be included in the materialization, however references to their immediate children would be included. This feature becomes quite valuable when working with large datasets, in that full documents do

not need to be materialized just to view partial results. It also allows one to build a demand driven Document Object Model (DOM) on top of the protocol. In general such a feature improves application performance by allowing the application to specify how data is materialized.

Querying

Mako provides query support through the use of both XPath and XQuery. XPath queries are performed using the XPathRequest packet. Similarly an XQuery can be performed using the XQueryRequest packet.

2.2.3 Exposing a Data Service with Mako

Data services can easily be exposed through the Mako protocol by creating an implementation of the abstract handlers. Since there is an abstract handler for each packet type in the protocol, data services can expose all or a subset of the Mako protocol. Once handler implementations exist, Mako can be easily configured to use them. This is done in the Mako configuration file, which contains a section for associating each packet type with a handler.

2.2.4 Global Addressing

We mentioned earlier that Mako provides a method of uniquely identifying elements contained in XML documents. In reality these elements are uniquely identified across the collection that they reside in. Mako also provides a method of globally addressing XML elements. This is done using the three tuple id, (Mako URI, collection, elementId). Being able to globally address entities within Mako provides several advantages; such as allowing data to be federated across multiple Makos.

2.2.5 Virtual Inclusion

In the last section we alluded to the fact that Mako allows data to be federated across multiple Makos, one example of this is virtual inclusion. Virtual inclusion, illustrated in Figure 7 is a reference within an XML document to another XML document. This means that Mako allows XML documents to be created that may contain references to existing XML documents or elements, both local and remote. In this way, an XML document can be distributed and stored across multiple Mako servers by storing subsections of the document remotely and integrating them with references. This ability is critical in enabling large data documents to be partitioned across a cluster while still maintaining the single document semantics of a model.

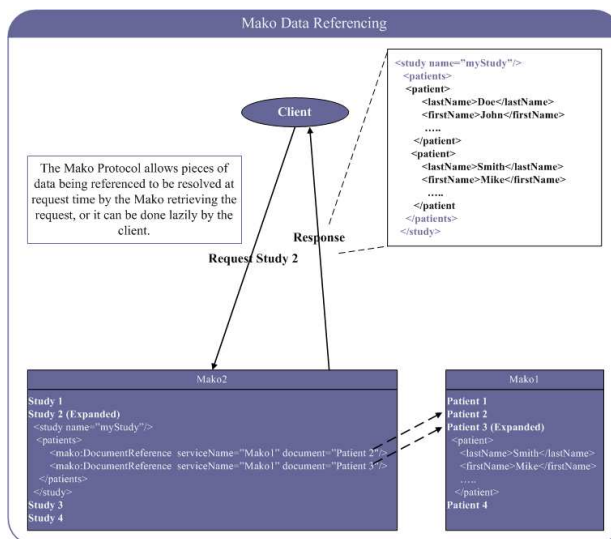


Figure 7. Mako Virtual Inclusion

2.2.6 Models

As mentioned in the earlier section, we designed some core models that any users can extend with specific versions which are more relevant to their domain. Figure 8 shows the basic single image type. The *basicImageType* is a simple data model which has four main elements: the image data or a pointer to it, a thumbnail, an id, and a generic set of key-value metadata. It also defines three required attributes: the image data encoding type, the height, and the width. This model is the basic starting point for representing an image in storage system. All images in the storage system will be or extend this type.

A simple extension to the 2D image model, shown in Figure 8, is the *volumeImageType* shown in Figure 9. The *volumeImageType* extends the *basicImageType* and adds a few more descriptive attributes which are relevant to the image if it belongs to a volume. The new attributes *xloc*, *yloc*, and *zloc* are attributes which describe the image starting location with respect to the volume container. Figure 10 shows the *volumeType* which is the container for a set of *volumeImageTypes*. This type contains a set of *volumeImageTypes* as well as some required metadata which describe the dimensions of the volume.

As mentioned above, our current application looks at the medical related imagery creating a PACS application with the purpose of handling medical image data. In order to accomplish this task we took the basic image and volume models described above, and used them in a new model which is more specific to medical image data sets. This model, shown in Figure 11, contains two basic elements, the image as defined by *basicImageType* or a volume as defined

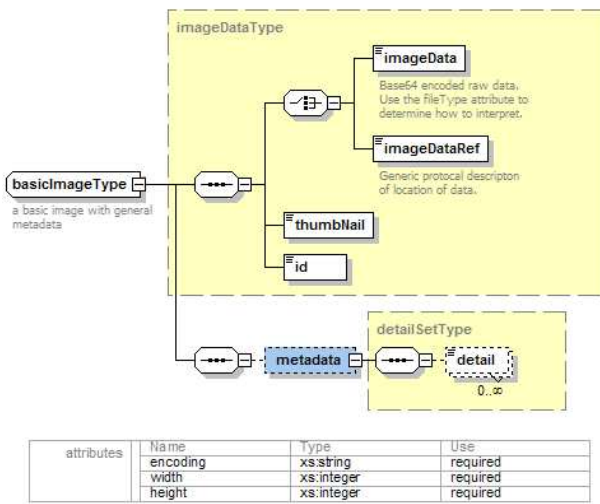


Figure 8. Basic Image Model

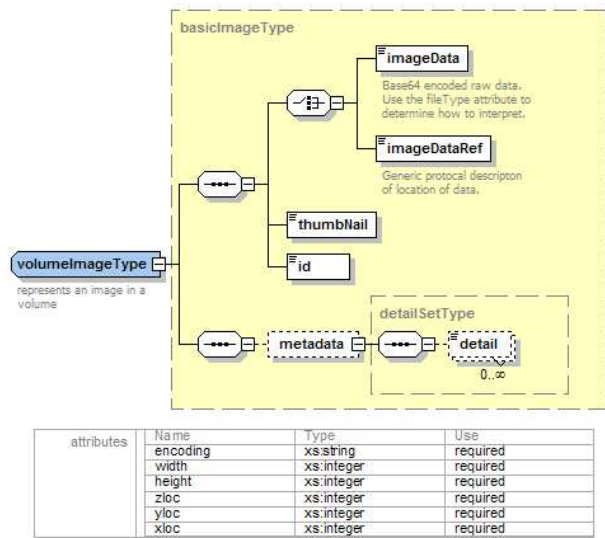


Figure 9. Volume Image Model

by *volumeType* and a set of medical image data set specific metadata. This model can be used to describe medical images or volumes and can be queried using the attached medical metadata and the metadata of the image data.

2.3 Test Architecture

Using the Mobius infrastructure and the models as described above we were able to create a distributed and scalable PACS system. We used two clusters for our basic test application. The first cluster is a five node dual Xeon, 320 GB disk, 2 GB ram, and dual gigabit ethernet network jacks each. The second cluster is a eight node dual 64 bit Opteron, 2 TB disk, 8 GB ram, and dual gigabit ethernet jacks. The two cluster are connected over a 100 Mb ethernet connection. We can place a MakoDB on each of the nodes and created a *PACS* collection. We then ingested the data models into each of the collection instances across the nodes. Now we have one large distributed model driven storage system.

To pre-load the PACS with test data we wrote command line ingestion tools that take existing medical data sets, create data model instances, and distribute them across the clusters. We initially ingested 140,000 images (approximately 50GB). Some of the images are in volumes and others are just stand alone data sets. Once the data was loaded, we were able to perform basic queries and aggregate data based on user defined metadata using the front end application.

Figure 12 shows a sample environment of front end clients and back end image data servers. The diagram shows that the Mobius middleware abstracts the location, storage medium, and data model from the clients. Clients can come

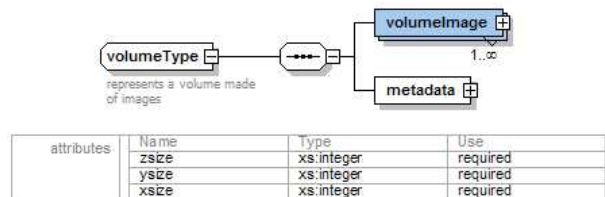


Figure 10. Volume Model

up and down from various places in the grid and make queries into the *image data grid*. Image data storage nodes can come up and down creating a dynamic ad-hoc environment of data services and clients.

3 Conclusion and Future Work

Overall this was a very successful project. We have shown that our framework is capable of scaling well and flexible enough to support user defined data models. We would like to stress it even more with more data and large and more complex queries. However, due to time and hardware constraints we were limited. The DPACS application is very useful as it currently stands. However, there are a few additions that we would like to add. Support for user defined viewer plug-ins would be a nice extension supporting image viewers which are tailored to handling the specific style of images in particular domains. We would also like to add a better query interface and provide for user extended query interfaces which could be tailored to the sci-

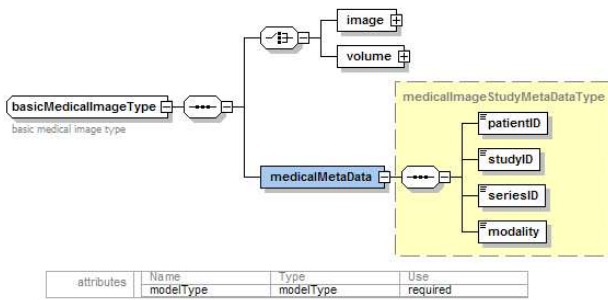


Figure 11. Medical Image Data Set Model

entists in their respective domains. This would make it easier to customize the front end application in order to better support the different user groups, while still leveraging the same back end distributed storage architecture. In the future we plan to begin working on these extensions, generalizing the architecture, and providing a stable and documented release to the community. The distributed image archival systems can then be customized and used by many different scientific communities. These communities will now be able to integrate their image data, mine it more effectively, and store quantities and sizes of data that were not previously feasible.

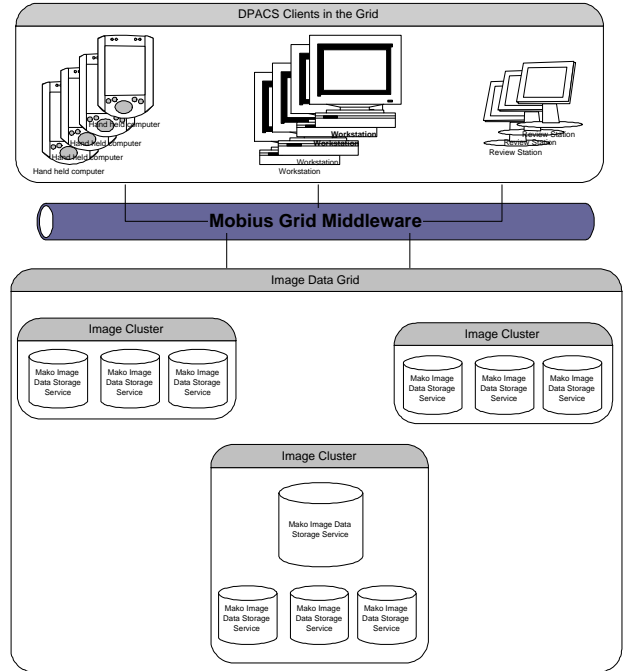


Figure 12. Sample Architecture