Data Synchronization and Replication Tool

Pradeeban Kathiravelu
INESC-ID Lisboa
Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal
Email: pradeeban.kathiravelu@tecnico.ulisboa.pt

Ashish Sharma
Department of Biomedical Informatics
Emory University
Atlanta, Georgia, USA
Email: ashish.sharma@emory.edu

***Abstract*—Consumers download the data by searching the image repository using the browser. The information that the consumer is interested in, gets updated whenever the data producers update or add patient information. The current download tool lacks the ability to track the relevant updates to the consumer. A data replication and synchronization tool will assist automated downloads to the consumers.**

I. Introduction

II. Background

Medical images are represented in multiple granularity. Figure 1 represents how the images are structured hierarchically in TCIA.

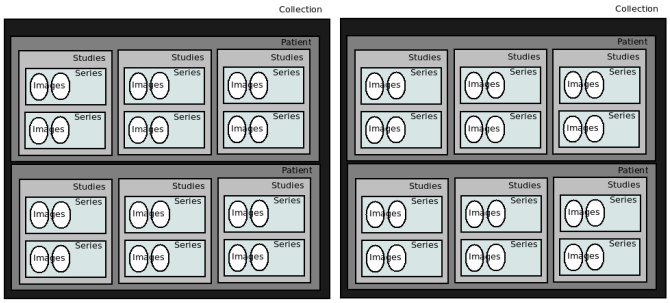

Fig. 1. Medical Images Granularity

III. Design and Implementation

A. Deployment

Having multiple instances running over different nodes provide fault-tolerance, as when one node terminates, the other nodes have the backup replica of the partitions stored in the terminated node. Figure 2 shows the higher level deployment view of the solution.

B. Design

Two distributed cache instances exist in InfDataAccessIntegration.

```
protected static Cache userReplicasMap;
protected static Cache replicaSetsMap;
```

userReplicasMap is a mapping of userId → Array of replicaSetIDs. UserID could be the logged in user name. (for now, testing with random strings). replicaSetsMap is a mapping of replicaSetID → replicaSet.

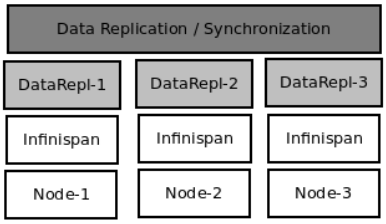

Fig. 2. Deployment

Though this could be replaced with a single cache instance with the mapping of userID → replicaSets, having two cache instances will be more efficient during searches, duplicates, and push changes. Hence, two cache instances design was chosen.

$InfDataAccessIntegration$ provides the API for publisher/consumer, $TCIAInvoker$ which extends the abstract class InterfaceManager, implements the TCIA integration to invoke these methods. Figure 3 provides a core class hierarchy of the system.

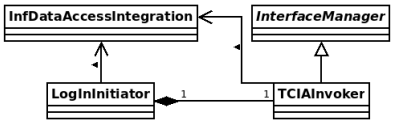

Fig. 3. Core Class Hierarchy

C. Execution Flow

The execution flow is depicted by Figure 4. When the user logs in, $logIn()$ checks whether the user has already stored replicaSets from the Infinispan distributed Cache. If so, execute them all again. This would be changed later as we do not have to execute all. Rather, we need to execute for the diffs. When the user performs new searches, for the images, series, collections, and the other meta data, the results will be returned to the user, and the user can chose ta subset of the returned results to create a replicaSet. The replicaSet for the image will be as,

```
TCIAConstants.IMAGE_TAG + "getImage?SeriesInstanceUID=" +
    seriesInstanceUID
```

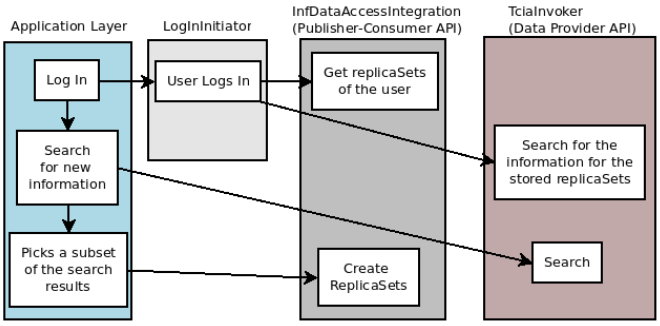

Fig. 4. Execution Flow

For other information (meta data), such as collections and seies,

```
TCIAConstants.META_TAG + query;
```

Here, query takes the below format.

```
"getSeries?format=" + format +
                "&Collection=" + collection +
                "&PatientID=" + patientID +
                "&StudyInstanceUID=" + studyInstanceUID +
                "&Modality=" + modality;
```

When a new instance starts now, and invokes the log in action for the same user, it will execute the queries for the stored replicaSets again, and reproduce the same results.

Figure 5 depict the methods that retrieve image and meta-data at different granularity from TCIA.

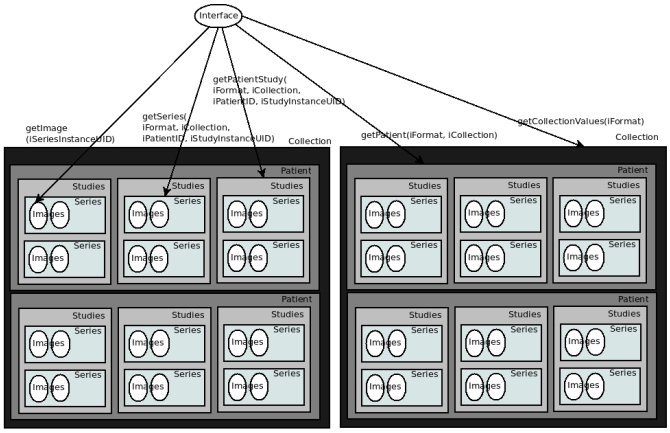

Fig. 5. Retrieving images and meta data

IV. Evaluation

V. Conclusion