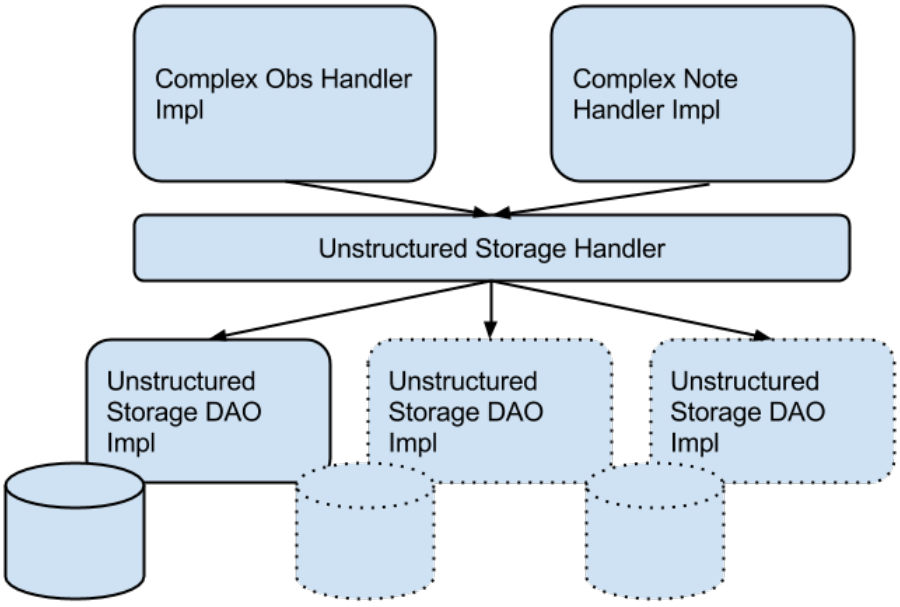
Unstructured Data Processing Module

This module will contain a Complex Observation Handler used to store unstructured data in the form of a complex observation to any storage systems suited for the purpose. The handler will be able to select a storage DAO according to the MIME type of the data that it's passed. So, for instance, a serialized object can be saved to a NoSQL database and large binary data can be saved to a distributed file/storage system.

It should be noted that there is no such thing in OpenMRS-core as a Complex Note as yet. We propose to get a ComplexData field included in the Note object and for the core to provide a ComplexNoteHandler interface. Any note with a non-null ComplexData property will be handled by a ComplexNoteHandler implementation.

Design



Implementing the ComplexObsHandler (and ComplexNoteHandler in the future)

Complex Obs Handler

```
package org.openmrs.module.shr.unstructureddata.obs.handler;

import java.io.IOException;
import org.openmrs.Obs;
import org.openmrs.api.APIException;
import org.openmrs.api.context.Context;
import org.openmrs.module.shr.unstructureddata.dao.handler.UnstructuredDAOHandler;
import org.openmrs.module.shr.unstructureddata.dao.handler.UnstructuredDAOService;
import org.openmrs.obs.ComplexData;
import org.openmrs.obs.ComplexObsHandler;
import org.openmrs.obs.handler.AbstractHandler;
public class UnstructuredDataHandler extends AbstractHandler implements        ComplexObsHandler {

        @Override
    public Obs saveObs(Obs obs) throws APIException {

        String contentType = getContentType(obs.getComplexData().getTitle());
        String key = obs.getUuid();

        if (Context.getService(UnstructuredDAOService.class).getUnstructuredDAO(contentType).saveObject(key,
obs.getComplexData().getData())){
            obs.setComplexData(null);
            obs.setValueComplex(obs.getComplexData().getTitle());
        }   else {
            throw new IOException();
        }

        return obs;
    }

    @Override
    public boolean purgeComplexData(Obs obs) {
        String contentType = getContentType(obs.getComplexData().getTitle());
        String key = obs.getUuid();
        return Context.getService(UnstructuredDAOService.class).getUnstructuredDAO(contentType).purgeObject
(key);
    }

    @Override
    public Obs getObs(Obs obs, String view) {

        String contentType = getContentType(obs.getComplexData().getTitle());
        String key = obs.getUuid();

              ComplexData complexData = new ComplexData(obs.getComplexData().getTitle(), Context.getService
(UnstructuredDAOService.class).getUnstructuredDAO(contentType).getObject(key));
              obs.setComplexData(complexData);
              return obs;
    }

    String getContentType(String title){
                return null;
    //do parsing here
    }
}
```

Service

Unstructured Data Service Interface

```
package org.openmrs.module.shr.unstructureddata.api;
import org.openmrs.api.OpenmrsService;
import org.openmrs.module.shr.unstructureddata.dao.UnstructuredDAO;
import org.openmrs.module.shr.unstructureddata.exception.AlreadyRegisteredException;

public interface UnstructuredDAOService extends OpenmrsService {
    /** To be called by the ComplexObsHandler/ComplexNoteHandler */
    UnstructuredDAO getUnstructuredDAO(String contentType);

    /** For each DAO a call to this method will be added in the ModuleActivator willStart() method */
    void RegisterUnstructuredDAO (String contentType, UnstructuredDAO prototype) throws
AlreadyRegisteredException;

    /** For each DAO a call to this method will be added in the ModuleActivator willStop() method */
    void DeregisterUnstructuredDAO(String contentType);
}
```

Interfaces

Unstructured DAO Interface

```
package org.openmrs.module.shr.unstructureddata.dao;

public interface UnstructuredDAO {

    Boolean saveObject(String key, Object value);

    Object getObject(String key);

    Boolean purgeObject(String key);
}
```

Possible Database / File Storage Options

As we suspect we'll be storing a lot of XML documents as blobs, a native XML database that supports XPath and XQuery is something we could be looking at.

BaseX seems to be the open-source XML database with the most freely available features, as shown by this comparison. We will have to decide whether to support XPath And XQuery on document-level data through the unstructured storage module.

For binary data storage a distributed file system seems like it will provide the best redundancy and scalabiltiy. RiakCS is a cloud storage platform that uses the Amazon S3 API and the Riak cluster module to provide high-availability storage. OpenStack Swift and Hadoop HDFS are other options to look at.

This blog post provides a good comparison of cloud storage architectures. As we are looking at solutions we can implement in our own data stores solutions like RiakCS, Ceph and OpenStack Swift are the main contenders for binary storage.

RiakCS provides easy distribution over multiple data centres and high availability. It has good documentation, and a widely used API and is released under the Apache2 licence.

THIS PAGE IS A WORK IN PROGRESS