FHIR: Cell-Level Security and Real Time Access with Accumulo
FHIR: Cell-Level Security and Real Time Access with Accumulo

by

Daniel Alfonso Ruiz

B.S, University of Colorado, Colorado Springs, 2008

A thesis submitted to Graduate Faculty of the
University of Colorado at Colorado Springs
in partial fulfillment of the
requirements for the degree of
Master of Science
Department of Computer Science
2015
This thesis for Masters of Science degree by

Daniel Alfonso Ruiz

has been approved for the

Department of Computer Science

by

_________________________
Dr. Rory Lewis
Thesis Director

_________________________
Dr. Edward Chow
Committee Member

_________________________
Dr. Terrance Boult
Committee Member

_________________________

Date
The American Recovery and Reinvestment Act (ARRA) requires the adoption of Electronic Medical Records (EMRs) for seventy percent of the primary care provider population by 2014. Furthermore, by 2015 providers are expected to be utilizing EHR in compliance with “meaningful use”[28] definition or they can face financial penalties under Medicare. In addition to this momentous task, EMR data has stringent security requirements. It is largely due to these security requirements that medical information is being digitized. However, sharing to entitled information is often slow or non-existent because of information silos. Fast Healthcare Interoperability Resources (FHIR) is an emerging information sharing standard that is designed to aid in the tearing down of these silos. The focus of this thesis is to show how FHIR can be further improved by allowing for cellular level security. Additionally, this thesis will introduce novel ways that vast amounts of FHIR resources can be stored and queried in real-time with Accumulo. It will do so by utilizing and improving on Dynamic Distributed Dimensional Data Model (D4M) [9] Schema to better allow for “real-time” REST queries of FHIR compliant data. Pagination is necessary for it to remain a real-time system since some queries can have millions or even billions of positive hits. To satisfy this requirement a new approach to Accumulo pagination is laid out that increases performance, flexibility and control. All tests are performed against a M4.2xlarge Amazon Machine Image.
ACKNOWLEDGEMENTS

First I would like to express my profound gratitude to my mother Amparo, brother David, wife Stephanie, daughter Jaylynn, and son Zavien, who’s support was instrumental in making this paper possible. I would also like to thank my dear friends Ryan, Irene and Grover for their support in helping make this paper possible. Finally, my thesis advisor Rory Lewis for always having his door open and being flexible with my busy schedule.
TABLE OF CONTENTS

1. Introduction ..................................................................................................................................................................................1
   1.1 Problem ....................................................................................................................................................................................2
   1.2 Objective ....................................................................................................................................................................................4
   1.3 Accomplishments .......................................................................................................................................................................4
   1.4 Organization of this Thesis ....................................................................................................................................................5

2. Background ..................................................................................................................................................................................6
   2.1 Definitions ..................................................................................................................................................................................6
      2.1.1 Accumulo ...........................................................................................................................................................................7
      2.1.2 Accumulo Setup ...............................................................................................................................................................8
      2.1.3 Redis .................................................................................................................................................................................10
   2.2 Related Work ............................................................................................................................................................................10
      2.2.1 Cell level Security with FHIR ........................................................................................................................................10
      2.2.2 Caching with Accumulo for iterator communication ..................................................................................................11
      2.2.3 Caching query state with Accumulo ................................................................................................................................11

3. Architecture ................................................................................................................................................................................13
   3.1 High Level Environment Architecture ..................................................................................................................................13
   3.2 Approach ...................................................................................................................................................................................14
      3.2.1 Scope ................................................................................................................................................................................14
      3.2.2 Cell Level Security Background ..................................................................................................................................14
      3.2.3 Cell Level Security Design ...........................................................................................................................................17
      3.2.4 Publish Design .................................................................................................................................................................20
LIST OF FIGURES

Figure 1 Pagination Code Example (Source: medined, Image by: David Medinets) ................... 12
Figure 2 Possible Resulting Output (Source: medined, Image by: David Medinets) ................... 12
Figure 3 Search for Page 3 (Source: medined, Image by: David Medinets) .......................... 12
Figure 4: FHIR Data Cloud Environment .............................................................................. 13
Figure 5 Ingest Design ............................................................................................................ 16
Figure 6 Generation and Publish process flow ...................................................................... 17
Figure 7 Adding secid Attribute to FHIR Schema .............................................................. 18
Figure 8 Publish Process ....................................................................................................... 21
Figure 9 Read Process .......................................................................................................... 24
Figure 10 Adding Cache to Accumulo .................................................................................. 28
Figure 11 Pagination with full resource results ................................................................. 29
Figure 12 Pagination with UUID results from cache .......................................................... 29
Figure 13 Sample Patient resource JSON message ............................................................ 30
Figure 14 Load Generator Publish Process ......................................................................... 31
Figure 15 Ingest Performance ............................................................................................. 32
Figure 16 Cell Level Security Demonstration ...................................................................... 33
Figure 17 Pagination Performance ...................................................................................... 34
Figure 18 Batch Query Performance ................................................................................... 35
1. INTRODUCTION

The American Recovery and Reinvestment Act (ARRA) required the adoption of Electronic Medical Records (EMRs) for seventy percent of the primary care provider population by 2014. Furthermore, by 2015 providers must utilize EMRs in compliance with “meaningful use”[28] definition or they can face financial penalties under Medicare. This change is bringing a significant shift in the medical community with respect to how they store data. With the shift to digital EMRs, now is the opportune time for the development and adoption of new standards that will define how data is shared for years to come.

Historically, healthcare has had many problems with sharing data. Information “silos”, databases that are difficult to share, can be found everywhere in healthcare. Hospitals, clinics and doctor’s offices are prime examples of where these silos are often found[37]. Since healthcare providers often use different software, an information sharing standard is needed to ease the process of sharing data[23]. Health Level Seven International (HL7) is the current standard use in the medical community to facilitate information sharing. However, HL7 is complex to understand and expensive to implement. Fast Healthcare Interoperability Resources (FHIR) is a new standard that is designed to ease the process of information sharing. Its goal is to reduce the cost and complexity by simplifying implementation without sacrificing information integrity.

The focus of this thesis is to show novel ways information can be shared using FHIR. It outlines the concept and benefits of adding cellular level security to FHIR. This will greatly improve the flexibility that data providers have when sharing FHIR resources. Moreover, it includes a concept demonstration and implementation referred to as the FHIR Data Cloud
(FHIR-DC) providing a great starting point for handling cellular level security with FHIR. The FHIR-DC can easily be scaled to handle vast amounts of information that can be stored and queried in real-time. To accomplish this, novel approaches for pagination along with optimizations to Accumulo using an In-Memory Database are introduced. The architecture also utilizes and provides preliminary support for the Dynamic Distributed Dimensional Data Model (D4M) [9] Schema. This brings the ability to quickly develop powerful analytics to FHIR data. The FHIR-DC also demonstrates a novel approach to pagination with Accumulo. For performance testing the FHIR-DC provides a generator capable of generating thousands of FHIR resources per second. Finally, it provides a test environment for both unit and integration tests that run against the database, which is hosted on two M4.2xlarge Amazon Machine Images.

1.1 Problem

FHIR is a data sharing standard. To facilitate this, each data resource within FHIR has one or many security codes that define how that resource can be shared. For example, a resource security code of U that indicates that the resource contains information that is not classified as sensitive and is therefore unrestricted [30]. The security code R specifies that data cannot be shared and is restricted. Individual data resources may contain both restricted and unrestricted elements. This means that data providers who want to share resources that contain both unrestricted and restricted information would need to publish two versions. One, with the resources as restricted and another with all the restricted information stripped. This process puts a lot of burden on the data providers along with the legal responsibility of ensuring that all the data was stripped correctly.
As a concrete example, assume a veterinarians opted to share information to the pharmaceutical industry about drugs they are prescribing. So they fill out the MedicationPrescription resource. However, they would also like to share information about the animal they are prescribing drugs to (i.e. species, breed, gender,...etc.). This information is contained in a Patient resource that they already have filled out. However, aside from the animal information the Patient resource also contains the owners contact information. Since the contact information, which is personally identifiable information, falls under a different set of sharing guidelines, the pharmaceutical companies do not have permission to access it. With security at the resource level, the pharmaceutical companies may not be able to access the information at all. They would have to have permission to view possibly restricted information. Despite the fact the part they really care about is generally unrestricted information, they will be denied the whole resource. In order to share that information the veterinarian either has to submit the MedicationPrescription resource with a copy of the patient information without the contact information or just simply not share that information.

Moving to a real-time system that will facilitate cellular level sharing to help avoid adding this additional burden on the data provider (i.e. the veterinarians) can be a significant challenge. At the very least, the following issues would need to be addressed:

- FHIR resource schemas would need to be redesigned or a complex information access policy (IAP) would need to be created to facilitate this type of sharing.
  - Each resources element would need to potentially be tagged with a specific security code.
- The system would need to have an IAP that facilitates cellular level access control.
The system would need to be able to handle the stripping of the data at a cellular level.

The system would need to handle access control in real-time. Nevertheless, without this type of capability FHIR can inadvertently limit the amount of data sharing due to the additional burden on the data provider to share certain types of information. Currently FHIR is in Draft Standard for Trial Use version 2 (DSTU2). Furthermore, most of the resources are considered to be at a low maturity level. Once it passes the trial use and becomes widely implemented, changing the schemas in such a drastic way would be practically impossible.

1.2 Objective

The overall objectives are to develop and demonstrate the first ever real-time (partial) FHIR implementation with cellular level security. To accomplish this, several innovative approaches of querying and indexing data on Apache Accumulo were developed and a novel approach to pagination.

1.3 Accomplishments

The accomplishments presented in this thesis lays the groundwork for more precise information sharing with FHIR data. It shows that having cell level security can be accomplished in real-time without the usual long and costly approach that Attribute Based Access Control (ABAC) usually entails. Moreover, it shows that this can be done in a potentially highly scalable and reliable environment. This lays the groundwork for integration
and later production level systems that can facilitate more precise sharing of medical data than is currently possible under the HL7 and FHIR DSTU2 standards.

1.4 Organization of this Thesis

Chapter 2 covers a background in FHIR, D4M and Accumulo, and the use of combining these technologies. Chapter 3 outlines the overall architecture and describes the difficulties in developing a high performance system with FHIR and Accumulo. It also outlines the improvements needed to D4M and current pagination approaches to deliver real-time performance for a larger set of queries. In Chapter 4, pagination performance and query performance is analyzed and compared to a non-cache approach. It also shows the overall performance and scalability of ingests and queries. Chapter 5 concludes the work with a summary and future work section.
## 2. BACKGROUND

### 2.1 Definitions

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute Based Access Control (ABAC)</td>
<td>Is a logical access control model that controls access to objects by evaluating rules against attributes of the entities actions and environment relevant to a request.</td>
</tr>
<tr>
<td>Amazon Web Service (AWS)</td>
<td>Amazon cloud computing environment.</td>
</tr>
<tr>
<td>AMI</td>
<td>Amazon Machine Image (AMI) is a template that contains a launchable software configuration of an instance.</td>
</tr>
<tr>
<td>Elastic Compute Cloud (EC2) Instance</td>
<td>A virtual server instance in AWS EC2 service.</td>
</tr>
<tr>
<td>Accumulo</td>
<td>NoSQL database based on Google’s BigTable design.</td>
</tr>
<tr>
<td>Cell Level Security</td>
<td>The capability to protect each field of data individually</td>
</tr>
<tr>
<td>Dynamic Distributed Dimensional Data Model (D4M)</td>
<td>Software developed by MIT focused on improving search, retrieval and analysis for Big Data applications.</td>
</tr>
<tr>
<td>Elastic Load Balancer</td>
<td>Part of the AWS EC2 service that automatically distributes incoming traffic across multiple EC2 instances.</td>
</tr>
<tr>
<td>FHIR-DC</td>
<td>A template implementation of cellular level access control with FHIR resource data.</td>
</tr>
<tr>
<td>Placement Group</td>
<td>A logical grouping of instances within an Availability Zone.</td>
</tr>
</tbody>
</table>
2.1.1 Accumulo

The release of Google’s Big Table[24] design in 2006, laid the foundation for designing and managing structured data on a large scale. With a focus on horizontal scaling, it allowed for storing and analyzing petabytes of information. These capabilities opened new doors of possibilities and help give birth to open source projects such as HBase, Cassandra, Hypertable and Accumulo. Implementing aspects of Google's Big Table design has allowed these open source projects to help revolutionize the way we look at data. They make it possible to analyze vast amounts of data often using only commodity servers.

Although HBase, Cassandra and Hypertable are also capable of handling vast amounts of information, Accumulo was chosen due to its ability to handle cellular level access control. Unlike HBase, which also has cell level security since version 0.98, Accumulo was designed around this feature and thus supported it since its conception. Another key feature of Accumulo is the server-side programming framework called iterators. Iterators are key to real-time performance with Accumulo and are heavily utilized in FHIR-DC. Iterators enable the ability to gain real-time query performance even when dealing with billions of rows despite being on a relatively low-performance environment. This is shown by using iterators along with Redis. This combination represents a novel approach to real-time performance with Accumulo, even on a tight budget.
Accumulo stores data on disk in the form of key-value pairs. These keys are guaranteed to be lexically sorted. Accumulo keys are made up of server components as shown in Table 1. Having several columns as the key allows for rows to contain multiple elements as opposed to each column mapped to a value in a relational database. Data is sent to Accumulo via a set of Mutation objects which contain a set of changes to be applied to a single row. Data can then be queried from Accumulo by iterating through key-value pairs.

**Table 1 Accumulo Key-Value**

<table>
<thead>
<tr>
<th>KEY</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>RowID</td>
<td></td>
</tr>
<tr>
<td>Column Family</td>
<td></td>
</tr>
<tr>
<td>Column Qualifier</td>
<td></td>
</tr>
<tr>
<td>Column Visibility</td>
<td></td>
</tr>
<tr>
<td>Timestamp</td>
<td></td>
</tr>
<tr>
<td>Value</td>
<td></td>
</tr>
</tbody>
</table>

2.1.2 Accumulo Setup

The FHIR-DC utilizes a two node Accumulo cluster that resides in AWS. The cluster consists of two M4.2xlarge instances based on the AWS provided Ubuntu Server 14.04 (ami-aeb532c6) AMI which is located in the same placement group. Both instance have a two General Purpose Elastic Block Store (EBS) presented to them, a 55GB SSD root volume and an additional 500GB SSD volume. The additional block device is mounted to /mnt/cachedata/. HDFS is setup to only utilize the 500GB SSD on /mnt/cachedata/. This setup helps increase performance by decoupling Accumulo and HDFS IO from OS and logging IO. For IO performance, AWS allocates 3 IOPS (input/output operation per seconds) per gigabyte [31]. Amazon EBS IOPS are measured as being 256KiB or smaller [32]. Additionally, there is currently a volume throughput max of 160 MiB/s [32]. Keeping this in mind along with the type
of load is important when considering a filesystem block size. For this thesis the default 4k blocks were used. With the default 4k blocks the FHIR-DC was able to demonstrate and meet all its performance objectives.

Accumulo was manually installed and configured on each box along with its dependencies Hadoop and Zookeeper. Currently, the versions being used are 1.7.0, 2.7.1 and 3.4.6 respectively. The Accumulo stack is set up on a two node cluster. One node contains all the master and slave processes. The other node contains only slave processes along with the In-Memory Database (Redis). Accumulo and Hadoop provide a large list of custom configuration that can be tweaked to improve performance. Since the effects of these tweaks are dependent on the system load, the defaults were generally used in this thesis. The configuration used for the FHIR-DC can be seen in Appendix A.

Several scripts were created to simplify starting and stopping Accumulo, Hadoop and Zookeeper. Since each software stack needs to be started individually and in a certain order, the scripts provide a single location that automates the stopping and starting of each component. To utilize the script simply log in as Hadoop user and run:

```bash
> sudo su hadoop
```

The start and stop scripts are found in /cloud/scripts. They can be executed by ./start-all.sh and ./stop-all.sh respectively. Both master and slaves must have the same configuration and libs to run correctly. This means configuration changes in the master need to be reflected on the slave node as well. To aid with this, there is a clusterUpdate.sh script that will copy any configuration changes on the master to the slave nodes. Lastly, to simplify adding new iterators to Accumulo,
a accumuloUpdate.sh script is available that stops updates and starts Accumulo with the new jars (which contain the new custom iterators).

2.1.3 Redis

Redis is an open source in-memory data structure store. Redis is similar to other technologies such as Memcached, however, Redis was chosen due to its tunable data persistence [33]. The version of Redis used for the FHIR-DC is 3.0.4. The configuration used for this version can be seen in Appendix B. For the FHIR-DC the current function of Redis is to act as a Cache for the Accumulo and query results for pagination.

2.2 Related Work

This thesis presents three novel ideas:

- Cell level security with FHIR
- Caching with Accumulo for internal iterator communication
- Caching query state with Accumulo

Sections 2.2.1 → 2.2.3 will discuss related work for each idea.

2.2.1 Cell level Security with FHIR

The idea of cell-level was first developed by Curt Gilroy and Roger Miller in 2008. They filed the patent under “Row-Level Security in a Relational Database Management System” Patent No: US 8,131,664 B2 and was granted in March 2012. Granular level security has been applied to different areas since its conception in 2008 on both relational and non-relational
databases. Details related to this topic are currently hard to find. Organizations, such as the
NSA, who developed Accumulo, are using cell level security but are not publishing the
information. This new technology of cell-level security has not yet been applied in many
domains. This thesis expands the state of the art by applying cell-level security to the medical
domain and FHIR data standard.

2.2.2 Caching with Accumulo for iterator communication

There are many different caching mechanisms in Big Table derived databases. For
example, HBase has BlockCache which significantly increase overall database performance [35].
There is also external caching such as SlabCache which can reduce the frequency of stop-the-
world (STW) pauses (i.e. JVM garbage collection) but not increase maximum pause time [36].
However, there are currently not any published projects that attempt adding external caching to
increase query performance by a service side mechanism such as filtering. This may not be
needed in HBase or Cassandra but with Accumulo this thesis makes the argument that it can be
very useful.

2.2.3 Caching query state with Accumulo

Unlike utilizing the external cache for iterator communication, the cache can also be used
to increase query performance by not having to query Accumulo. A great example that
demonstrates the power of this idea can be seen with pagination. As explained in “Pagination
with Accumulo” [36], pagination with Accumulo is not simple. The pagination method laid out
in the article shows how selectable pages can be achieved with Accumulo. As shown in Figure
1, the total pages can be calculated at query time. In practice (though not shown in this example) 
fieldName will be saved off somewhere. Assuming fieldname is saved off to a map from Figure 
2, the key would be 03 and the value would be a10300. Then if a user requested page 3 a scan 
with range starting at a10300 would be performed (see Figure 3).

```
scanner = connector.createScanner(tableName, new Authorizations());
Iterator<Map.Entry<Key, Value>> iterator = scanner.iterator();
while (iterator.hasNext()) {
    Entry<Key, Value> entry = iterator.next();
    String fieldName = entry.getKey().getRow();
    if (entryCount == 0 || (entryCount % pageSize == 0)) {
        System.out.println(String.format("%d,%d", pageNumber, fieldName));
        pageNumber++;
    }
    entryCount++;
}
```

Figure 1 Pagination Code Example (Source: medined, Image by: David Medinets)

| 1, a00100 |
| 2, a02500 |
| 3, a10300 |
| 4, a59660 |
| 5, n0100 |
| 6, n07220 |
| 7, n18450 |
| 8, state |

Figure 2 Possible Resulting Output (Source: medined, Image by: David Medinets)

```
scan.setBatchSize(batchSize);
scan.setRange(new Range(new Text("a10300"), true, null, true));
```

Figure 3 Search for Page 3 (Source: medined, Image by: David Medinets)

There are some issues with this approach as pointed out with in the article but in general this is 
how pagination is currently approached in Accumulo.
3. ARCHITECTURE

3.1 High Level Environment Architecture

The FHIR-DC is hosted in AWS. The functionality is separated into several AMI instances to allow for horizontal scaling shown in Figure 4. The environment consists of a Tomcat server that hosts REST web service. To allow for scaling the web service is behind an ELB. Accumulo sits on top of two AWS M4.2xlarge instance types. One instance contains the master and slave process while the other contains only slave processes and Redis. The Load Generator in hosted on a C4.xlarge instance. To ensure the highest possible bandwidth, all the services hosted in AWS are in the same Placement Group. A System Test Environment is currently located on the local development box outside of AWS.

![Figure 4: FHIR Data Cloud Environment](image_url)
3.2 Approach

3.2.1 Scope

The focus of the FHIR-DC is to demonstrate real-time cellular level access control with FHIR. FHIR-DC is not a full implementation of FHIR, but rather aims to be a starting point for an integration and possibly even a production level implementation. Creating a complete FHIR implementation will take a considerable amount of time. However, the FHIR-DC is meant to provide a strong starting point. To enable this, future plans are to make it an open source project available through GitHub.

Building a complete implementation of FHIR resources, REST API, web level security (i.e. SSL), user security handling (i.e. how users will be assigned credentials), system security handling, etc., is not needed to demonstrate that real-time cellular level security is possible or how it can be achieved. For example, the performance consideration of SSL is negligible when compared to the process involved with satisfying a FHIR query. Furthermore, since resources share the same types (i.e. Domain Resource, Boolean, Data...etc.) and have similar schemas, demonstrating with two resources is enough to give a high level of confidence that it will work with many resources.

3.2.2 Cell Level Security Background

The focus of the FHIR-DC architecture is to fulfill its role as a concept demonstrator. FHIR-DC supports cell level security with FHIR resources, however FHIR does not yet support
cell level security in its schema. As such, the decision was made to avoid using custom FHIR schemas and schema based objects but to use what is available and suggested in the community. This significantly reduces the amount of work needed to keep FHIR-DC current as new releases roll out. This is especially important due to most resources being in a low maturity level and therefore likely to have changes. Rather than changing the core schemas and objects, the FHIR-DC simulates cell level security. The FHIR-DC is fully compatible with both cell level and resource level security models, so if needed the cell level can be easily turned off or on.

The publish feature in FHIR-DC is designed to decouple the creation of the mutation from the schema based objects. As seen in Figure 5 the schema based resource objects are not directly converted into mutation. Instead mutation creation relies on table packet objects. Therefore, as long as the table packet stays consistent (which it is fully expected to) the table structure and contents won’t change. This is true even if the schema based resources where to adopt cell level security or go through a major overhaul.
To simulate cell level security, the information that would be contained in the secid attribute value per proposed schema change is inserted at the table packet layer. When the table packet is created it contains the same information for cell level security as it would, if it were actually part of the resource object. Thus not having the secid attribute as part of the resource object makes no difference with respect to the column visibility field of the mutation. As far as the mutations and tables are concerned there is cellular level security. From a performance standpoint, the amount of time to access the attribute vice artificially inserting is negligible, even when handling thousands of records per second.
3.2.3 Cell Level Security Design

Adding cell level security to FHIR can be done with minimal schema changes from the current DSTU2 version. Cell level security proposed in this thesis uses the current resource level security infrastructure and business rules. From a schema level, the main addition is a secid attribute to each element (see Figure 7). Although making this change has the desired effect of adding cell-based security to each element, there may be some fine tuning that is needed. For example, there are some elements, such as coding, where system and code elements should be separated and therefore should always have the same secid value. Without understanding the FHIR environments and context for which it is used, it is difficult to say what fields we may not want to have a secid extension. A possible alternative to updating the xsd’s to accommodate any restriction is to have restrictions of secid attribute usage as business rules rather than enforced at
the schema level. This follows the current pattern of FHIR in allowing practically every element
to be optional and relying on business rules to ensure compliance with the standard.

Figure 7 Adding secid Attribute to FHIR Schema

The following is an example taken from the FHIR website (https://www.hl7.org/fhir/) and
having the secid attributes added in:

U,PDS,GENDER

```
  <id value="animal"/>
  <meta>
    <security id="SI1ff2">
      <system value="http://hl7.org/fhir/v3/Confidentiality"/>
      <code value="U"/>
    </security>
    <security id="SI2AB2">
      <system value="http://hl7.org/fhir/v3/ActCode"/>
      <code value="PDS"/>
    </security>
    <security id="SI3DP2">
      <system value="http://hl7.org/fhir/v3/ActCode"/>
      <code value="GENDER"/>
    </security>
  </meta>
  <!- Done at block level all child elements inherit specified secid -->
  <!- Dog tag, under Maroondah City council -->
  <identifier secid="SI1ff2">
    <type>
      <text value="Dog Tag"/>
    </type>
    <value value="1234123"/>
  </identifier>
</Patient>
```
Additional business rules may be needed to govern inheritance. For example, it may be easier with the DS4P CDA xml based specification to label a block with one secid attribute and have all child elements inherit that value. Another case is when you want an element to have more than one attribute. Since, as defined in the schema, there should only be one code per Coding
element, it may be best to allow for the secid to represent multiple codes. For instance, the secid could reference two or more security codes with a comma delimited list i.e. 
secid=“SI2AB2,SI3DP”.

3.2.4 Publish Design

Currently the FHIR-DC only supports publishing of FHIR resources in JavaScript Object Notation (JSON) format. This is due to the simplicity of working with JSON and both where not needed to solidify the concept implementation. HL7 provides and open source java library for FHIR development. The library is available through the Maven Repository (http://mvnrepository.com/artifact/me.fhir/fhir-dstu2/1.0.1.7108) under the groupId of me.mfhir and artifactId of fhir-dstu2. For the FHIR-DC the version that was used is 1.0.1.7108. Among the useful tools in the library are Java objects for each supported resource, as well as parsers for eXtensible Markup Language (XML) and JSON formats. Using these Java objects with cell level security provides the best performance of extracting FHIR resource elements when compared to iterating through JSON data.

Every resource that is publish is marked with an internally generated universally unique identifier (UUID). Any updates are also marked with a newly generated internal UUID. The UUID is globally unique with the FHIR-DC and is used to ensure that indexes retrieve, not only the respective record, but the same instance of that record. The publish functionality also needs to be able to scale to ingest thousands of resources per second. To do this, it is important to utilize the resource object when extracting each individual element (known as “shredding”) in
the case where cell level security is needed. Note, that resources are only completely shredded when there is more than one security id.

![Figure 8 Publish Process](image)

**Figure 8 Publish Process**

### 3.2.5 Record Table Design

The record table (RecordTable) contains all the publish resource information in its entirety. To support cellular level security, there are two case scenarios for storing the full text of the resource. For both use cases, the assumption is that updates to records would republish the entire record with the updated information. This has the advantage of ensuring there is no data lost and extensive history of that particular resource id is kept intact. If updates of certain fields where allowed, then history would have to rely on Accumulo’s versioning feature, but this makes receiving and understanding the entire history of a record less straightforward and more prone to
errors. Another issue with this approach is ensuring indexes from edge tables remain consistent. Updating a delta on a record table is straightforward, but updating all references to reflect that specific change is non-trivial and far costlier (when compared to a republish).

➢ **Use Case 1:**

There is only one security code that is utilized or defined throughout the resource. For this case every element in the resource has the same security requirements and therefore the resource does not need to be shredded. Instead, it is compressed and ingested in its entirety in a single mutation. To reconstruct the resource at query time all that is needed is the entitlement level (e.g. resource level) and the resource type (i.e. Patient, Location, Medication…etc.). The entitlement level indicates which use case the mutation belongs too, while the resource type ensures correct casting of the message into a resource object. This information is stored in the column qualifier and is compliant with the D4M TedgeText type table.

| **Table 2 Record table layout for use case 1** |
|---|---|---|---|---|---|
| **RowId** | **Column Family** | **Column Qualifier** | **CV** | **Timestamp** | **Value** |
| UUID_1 | | EntitlementLevel|0| ResourceType|0| Required Entitlements | Ingest Time | <FULL RESOURCE INSTANCE> |

➢ **Use Case 2:**

There are multiple security codes that are defined and utilized throughout the resource. In this case the resource needs to be completely shredded. Each element needs to have their corresponding security codes attached when put into a mutation. Moreover, the aggregate of the mutations needs to contain enough information to reconstruct the resource at time of query. This
requires additional information added to the column qualifier that is described in Use Case 1.

The additional information can be summed up as an xpath and node type which indicates the type of 1st level root node child. For convenience the xpath is broken up into two parts, the id and the node name path. As an example, the xpath patient[0]/address[0]/city[0] and patient[0]/address[1]/city[0] will have the node ids of 0|0|0 and 0|1|0 and the same node path of address/city respectively. The node type for both examples would be address_list. Since every element within the respective resource has the same UUID and the same value for EntitlementLevel, Accumulo guarantees they will be in lexicographically ascending order based on the node id. This guarantee and ingest pattern simplifies reconstructing the resource in the same order that it was ingested. If an element does not have a security id defined, only the value is returned, otherwise the security id followed by the element value is returned.

Table 3 Record Table layout for use case 2

<table>
<thead>
<tr>
<th>RowId</th>
<th>Column Family</th>
<th>Column Qualifier</th>
<th>CV</th>
<th>Timestamp</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UUID_1</td>
<td>EntitlementLevel:0</td>
<td>NodeId:0</td>
<td>ResourceType:0</td>
<td>NodeType:0</td>
<td>Node Path(0 represents null and is used for parsing)</td>
</tr>
<tr>
<td>UUID_2</td>
<td>EntitlementLevel:0</td>
<td>NodeId:0</td>
<td>ResourceType:0</td>
<td>NodeType:0</td>
<td>Node Path(0 represents null and is used for parsing)</td>
</tr>
<tr>
<td>UUID_3</td>
<td>EntitlementLevel:0</td>
<td>NodeId:0</td>
<td>ResourceType:0</td>
<td>NodeType:0</td>
<td>Node Path(0 represents null and is used for parsing)</td>
</tr>
<tr>
<td>UUID_4</td>
<td>ResourceLevel:0</td>
<td>ResourceType:index</td>
<td>path</td>
<td>Req. Element Entitlements</td>
<td>Ingest Time</td>
</tr>
<tr>
<td>UUID_5</td>
<td>ResourceLevel:0</td>
<td>ResourceType:index</td>
<td>path</td>
<td>Req. Element Entitlements</td>
<td>Ingest Time</td>
</tr>
<tr>
<td>UUID_6</td>
<td>EntitlementLevel:0</td>
<td>NodeId:0</td>
<td>ResourceType:0</td>
<td>NodeType:0</td>
<td>Node Path(0 represents null and is used for parsing)</td>
</tr>
</tbody>
</table>
A resource when republished with updates over time can feasibly have instances that are part of use case 1, and other instances that are part of use case 2. Utilizing both use cases happens seamlessly in the read request process that is shown in Error! Reference source not found.

![Figure 9 Read Process](image)

### 3.2.6 Edge Tables Design

Given the record id, the UUID Edge Table (Table 4 UUID Edge Table) allows for O(1) access. The message time is used to keep the messages in lexicographical ascending order. To ensure this order the latest date needs to be to smallest value. The message time needs to be
adjusted in a similar fashion to 798487867887, this can be calculate as follows (note time is the format year:month:day:hour:minute):

- 999999999999 - 201512132112 = 798487867887

This table is currently used during the read process as shown in Figure 9 Read Process.

### Table 4 UUID Edge Table

<table>
<thead>
<tr>
<th>RowId</th>
<th>Column Name</th>
<th>Column Qualifier</th>
<th>CV</th>
<th>Timestamp</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RecordId</td>
<td>MessageTime</td>
<td>UUID</td>
<td>Required Entitlements</td>
<td>Ingest Time</td>
<td></td>
</tr>
</tbody>
</table>

The LeafTable allows for O(1) access to a unique leaf/value combination. If the leaf/value combination is abundant like country/us then the access time is O(n). When dealing with millions or even billions of rows the O(n) can take longer than what is needed for real-time access. To assist in this problem, the message time (msgTime) is added for the following reasons:

1. Keep the leaf/value pair in lexicographical ascending order
2. Allow for queries to specify a data time range which can drastically reduce the number of positive hits.
3. Reduce time for pagination as seen in Figure 17 Pagination Performance

This table is used in search queries as seen in Figure 11 Pagination with full resource results.
Table 5 LeafTable

<table>
<thead>
<tr>
<th>RowId</th>
<th>Column Family</th>
<th>Column Qualifier</th>
<th>CV</th>
<th>Timestamp</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LeafPath</td>
<td>LeafValue</td>
<td>msgTime</td>
<td>UUID</td>
<td>Required Entitlements</td>
<td>Ingest Time</td>
</tr>
</tbody>
</table>

The TLeafTable is a transpose table used to keep compliance with D4M schema and is used to allow for O(1) access time to a set of leaf/value pairs given a specific UUID (This table is not currently being used in FHIR-DC).

Table 6 TLeafTable

<table>
<thead>
<tr>
<th>RowId</th>
<th>Column Family</th>
<th>Column Qualifier</th>
<th>CV</th>
<th>Timestamp</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>UUID</td>
<td></td>
<td>LeafPath</td>
<td>LeafValue</td>
<td>Required Entitlements</td>
<td>Ingest Time</td>
</tr>
</tbody>
</table>

The LeafDegreeTable is a used to keep compliance with D4M schema. Its purpose it to allow provide counts for each leaf/value pair. This table uses an Accumulo provided aggregating iterator to update the counts in the value column.

Table 7 LeafDegreeTable

<table>
<thead>
<tr>
<th>RowId</th>
<th>Column Family</th>
<th>Column Qualifier</th>
<th>CV</th>
<th>Timestamp</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>leafPath</td>
<td>leafValue</td>
<td></td>
<td>Ingest Time</td>
<td>Count</td>
<td></td>
</tr>
</tbody>
</table>

The LeafFieldTable is used to keep compliance with the D4M schema. Its purpose is to allow provide counts for each leaf. This table also uses an aggregating iterator to update the counts in the value column.
### 3.2.7 Optimizations with In-Memory Database

By design Accumulo is a stateless system. Being a stateless system simplifies horizontal scaling and fault tolerance. Due to its stateless nature server side iterators are unable to share data.

Since a scan using a custom iterator will typically go through multiple instances of that iterator during the scanning process, global variables are not very reliable. This is due to new instances of the specified custom iterator being spun up and re-initializing. The FHIR-DC solves this issue by keeping the iterator state in an external Cache (see Figure 10). Adding iterator state solves this issue by allowing for the ability of iterators to effectively have dependable global variables. It also allows for iterator communication which is a novel approach that provides a great deal of additional flexibility when designing Accumulo iterators. Since the state is external, new algorithms can be created that reliably leverage information sharing to create reliable global variables, or even handle multiple iterators from the same or even different tables. Additionally, the cache can also be used decrease query time by caching query results. This is particularly useful when handling pagination or complex queries that have longer than average query times or just to reduce the overall load on the system.
3.2.7.1 Pagination

As discussed in section 2.2.3 pagination is a difficult problem to solve with Accumulo. The FHIR-DC takes a novel approach to this problem by utilizing the external cache. The cache utilization for pagination is described in two scenarios:

- Scenario 1: Pagination with full resource results
  
  In this scenario the search request is process and a hashed key
Scenario 2:

Figure 11 Pagination with full resource results

Figure 12 Pagination with UUID results from cache
3.3 Message Generation

The FHIR-DC message generator is designed to generate queryable information for testing purposes. A fake data generator (https://github.com/Codearte/jfairy) is used to generate personable information. The generator also uses a ZIP code data file to ensure realistic city, state, and ZIP code information. An example of a Patient resource generated with the generator can be seen in Figure 13.

```
{ "resourceType": "Patient", "id": "271def5f-eb84-4ebf-8be5-6059f6f5f26", "meta": { "security": [ { "id": "U_516d0d10-fc1f-4b0e-8a58-962c7cd947", "system": "http://hl7.org/fhir/valueset-security-labels.html", "code": "U" } ] }, "text": { "status": "generated" }, "identifier": [ { "system": "urn:oid:8.0.9;89.607", "value": "0941977" } ], "active": true, "name": [ { "use": "official", "text": "David Coleman", "family": [ "Coleman" ], "given": [ "Lori" ] }, { "telecom": [ { "system": "phone", "value": "946-196-2786", "use": "work" } ] }, { "gender": "male", "birthDate": "1992-05-02", "deceasedBoolean": false, "address": [ { "use": "home", "line": [ "99 truth and who" ], "city": "TULUCA LAKE", "state": "CA", "postalCode": "91610", "country": "US" }, { "use": "old", "line": [ "67 occur it right" ], "city": "STOCKDALE", "state": "TX", "postalCode": "78160", "country": "US" } ], "contact": [ { "name": [ { "use": "official", "text": "David Coleman", "family": [ "Coleman" ] }, { "given": [ "Moreno" ] }, { "address": [ { "use": "home", "line": [ "23 why but ever" ], "city": "TOBACCOVILLE", "state": "NC", "postalCode": "27050", "country": "US" } ], "gender": "male" ] } ] }
```

Figure 13 Sample Patient resource JSON message

3.4 Load Generation

The FHIR-DC has a load generator to generate FHIR resources in high volumes to test throughput and to rapidly increase table size. It does so by parallelizing the creation of randomly
generated FHIR resources and writing those resources directly to Accumulo. The process of the load generation as shown in Figure 14. A CLI is used to govern the amount of threads, batch size and total message count of the messages that will be published.

Figure 14 Load Generator Publish Process
4. METHODOLOGY, RESULTS AND ANALYSIS (OR SIMILAR TITLE)

4.1 Methodology

The FHIR-DC has a testing framework (FdcCorroborator) based on TestNG to test functionality and design, and a load generator (FdcLoadGenerator) to test performance. The FdcCorroborator consists of several tests that allow for testing of core FHIR-DC system functionalities. The tests are designed to test the functionality and performance of batch publish, batch query and pagination.

4.2 Results

![Ingest Performance](image)

**Figure 15 Ingest Performance**
Ingesting is parsing input data down to the element level and storing them in the Accumulo database. Figure 15 shows that the system can ingest at a rate of 10 MB/s, which is excellent performance for a two-node cluster with suboptimal configuration. At this rate, it was able to ingest 150,000 records per second.
Figure 16 shows demonstrates the JSON output of FHIR-DC, which includes cellular security. The left column shows the resource as viewed by a client that has authorization to view employment related attributes. The right column shows the resource as viewed by a client that does not have these authorizations. As you can see, much of the data is automatically redacted.

![Paginaiton Performance](chart)

**Figure 17 Pagination Performance**

The chart in Figure 17 shows the difference between the standard D4M and the approach using a timestamp in the rows. It also shows the effect of caching. The pagination implementations have a faster query performance than the standard D4M, caching also increases query performance slightly.
Performance scales linearly with page size as seen in Figure 18 for the tests that were run. This shows that it can perform as a real-time system with reasonable sizes of batches.
5. CONCLUSIONS

5.1 Summary

Cellular level security can greatly benefit the FHIR standard by allowing for flexibility of data providers and more information for data consumers. It also enables more system-negotiated interactions between parties for sharing data while preserving appropriate privacy guards.

This flexibility allows for data sharing under emergency circumstances without the need for new technologies or agreements to be developed.

This thesis also shows that this functionality can be achieved in real-time, thereby opening new possibilities for rapid response to epidemics and other disasters as well as rapid response during individual care in the healthcare system.

5.2 Significance

The FHIR-DC brings cell level security to FHIR, making strides to solve the problem of security, privacy and data sharing in the medical field. Furthermore, it acts as a proof of concept that shows it can be done in real-time. As laid out throughout this thesis, adding cell level security to FHIR will greatly increase the ability to share information in healthcare. This feature with encourage conversations on what elements can be shared and which resources should be shared. By providing security at a granular level, this enables for more information to be shared. Instead of deciding at the resource level what can be shared, sharing can be discussed at each individual element, thereby allowing details to be shared while maintaining privacy.
5.3 Future Work

As a proof of concept there is significant amount of future work that needs to be done in order to become a production level system. However, there is also more “template” work that needs to be done as well. To better solidify FHIR-DC the following features still need to be completed:

- Create each FHIR resource for both Publish and Query.
- Create a backend API layer to allow for support of both SQL and NoSQL systems

As well as other items that may be identified as work continues

In addition, some more work and research for the appropriate backend may be needed. Accumulo appears to be a good choice for a NoSQL approach makes given the current available technologies. It also may make sense to use both a SQL and NoSQL hybrid approach dependent on the query and context. Thus future work would include showing the feasibility of this approach and how it can benefit the overall architecture.
REFERENCES


[29] FHIR overview. [https://www.hl7.org/fhir/overview.html](https://www.hl7.org/fhir/overview.html)


[36] D4M pagination with
https://github.com/medined/D4M_Schema/blob/master/docs/pagination.md

APPENDIX A.

The following shows that the modified Patient resource xml validates after schema change to support cell level security.

Full example of FHIR JSON Patient resource with cellular level security.

```json
{
    "resourceType": "Patient",
    "id": "e773f2e6-c4dc-4704-a4de-d84da00634c7",
    "meta": {
        "security": [
            {
                "id": "52a09cd5cffd479cbdedc7840d9cd17c",
                "system": "http",
                "code": "V"
            },
            {
                "id": "edf7ebca807c4d5b93addf6cd70779ad",
                "system": "http",
                "code": "EMPL"
            }
        ]
    }
}
```
Avery Cook