A PROJECT REPORT ON

DISTRIBUTED HEALTH RECORD MANAGEMENT FOR HETEROGENEOUS HOSPITAL INFORMATION SYSTEMS

Submitted in partial fulfillment of the requirement for the award of the degree of

Master of Technology in Software Engineering

Submitted By:

Arpit Goel 2K13/SWE/02

Under the guidance of

Dr. Daya Gupta

Professor

Co-guide

Ms. Divyashikha Sethia

Asst. Professor

DELHI TECHNOLOGICAL UNIVERSITY
(Main Bawana Road, Shahabad Daulatpur, New Delhi, Delhi 110042)

(2013-2015)

CERTIFICATE

This is to certify that the project entitled **"Distributed Health Record Management for Heterogeneous Hospital Information Systems"** is being submitted at DTU, Delhi for the award of **Master of Technology** in **Software Engineering** degree. It contains the record of bonafide work carried out by **Arpit Goel** under my supervision and guidance. It is further certified that the work presented here has reached the standard of M.Tech and to the best of my knowledge has not been submitted anywhere else for the award of any other degree or diploma.

Dr. Daya Gupta
(Professor)
Department of Software Engineering
DTU, Delhi

ACKNOWLEDGEMENT

I feel immense pleasure to express my heartfelt gratitude to Ms. Divyashikha Sethia for her constant and consistent inspiring guidance and utmost co-operation at every stage which culminated in successful completion of my project work.

I am also grateful for Dr. Daya Gupta's overall supervision and direction in the research work. I also would like to thank the faculty of Computer Engineering Department, DTU and my peers for their kind advice and help from time to time.

I owe my profound gratitude to my family which has been a constant source of inspiration and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

A patient can visit numerous hospitals or clinics in his lifetime which therefore disseminates his medical data among multiple hospital information systems (HIS), managed by each hospital independently. The integrated Patient Health Records can be kept on portable equipment such as USB or mobile device [1], and can be retained with the patient as he moves from one HIS to another. A backup of the PHR should be retained on the cloud for refurbishing PHR in case of theft or for a remote access. The proposed system on the cloud, shall maintain backup of healthcard consisting of references to each HIS having patient's actual clinical data along with medical information like medication, lab tests. To integrate the dispersed data, we propose a distributed health record management system that shall query individual HISs for clinical data of a particular patient and then integrate all the responses together in standardized way to generate the aggregated health record in case of loss of PHR based healthcard. The information we receive in response from diverse sources may all be in different formats. The proposed system shall translate all responses into HL7 messages that together may represent the integrated Electronic Health Record (EHR) of the patient. We have deployed the backup of records on a distributed file system such as Hadoop DFS and results indicate that it can provide high availability, reliability and performance as compared to a single file system access.

CHAPTER 1: INTRODUCTION

A Personal Health Record (PHR) is loosely defined as a patient data repository that can serve individuals, encouraging self-care. It allows an individual to keep track of his complete medical history including current and past diagnoses, medications, lab results and much more [2]. PHRs can also be used for providing efficient communication between patients and providers and even allow patients to setup future appointments [3]. It enables consolidation of medical data from patients as well as physicians. Patient collected information may include various physiological parameters that can be gathered through body sensors and physicians may specify clinical information such as lab tests results, prescribed medication or identified allergies. Apart from clinical information, PHRs can also store other information like patient's insurance information for billing purposes and emergency information that can be used in crisis or during disasters [4]. They can be extremely valuable for chronic patients in managing their long term illness.

There are various types of PHRs that are being implemented today. In a broad view, they can be classified as either standalone PHRs or integrated PHRs [4, 5]. *Standalone PHRs* are generally under the physical control of the individual and they can be carried on devices like USB drives or smart cards. The patient himself may create and update these records. Such PHRs can provide portability, security and confidentiality of records but the information cannot be trusted by a healthcare provider as patients may not be able to report exact test results making them inadequate [6]. *Integrated PHRs,* on the other hand, include records from various sources such as Hospital Information Systems (HIS), pharmacies or insurance companies where patient data is generally kept in a centralized repository and made available to patient through web or email. An integration solution offers more complete records that can be trusted by providers. Although, integrated PHRs have huge potential, they are difficult to realize in practice [4, 5]. To reap benefits from both architectural styles, a hybrid PHR can be used which consists of a patient controlled health record, termed as healthcard, containing patient's vitals, insurance information, clinical observation, lab reports, prescriptions and medications. The healthcard, as outlined in Fig. 1, can exchange data with various healthcare stakeholders such as doctor, nurse, pharmacist, lab technician etc. in a standardized manner, promoting both self-care and communication among patients and healthcare providers [7].

Medical information for patients can be distributed among numerous healthcare facilities as the patient may move between different hospitals to seek specialized treatment or secondary consultation. In certain situations, patient data may be tethered to an insurance plan or a particular facility but he may move from one medical plan to another, disseminating the medical records. Although, the decentralization of records may deter healthcare services due to unavailability of complete medical account, it still outweighs a centralized strategy as it raises many legal, security and privacy concerns. The NPfIT system that has been developed in U.K uses Spine to hold the complete patient record centrally [8]. The system has met with severe criticism regarding confidentiality and misuse of patient data, summarized in [9]. In contrast, Netherlands developed an Electronic Patient Dossier (EPD) system, that decentrally stores patient records into GBZs which are information systems maintained by each facility individually [10, 11]. Fragmented records do help in alleviating confidentiality issues but it must deal with the incorporation of dispersed data and keeping it in a standard format.

The work assumes that patient retains PHR on a portable device such as a mobile device [1] and proposes an integrated health record management solution in which records are decentrally stored on different HIS. Each record stored in HIS usually represents a visit or an encounter with a healthcare professional. A backup of the healthcard is retained on the cloud in a digital health vault consisting of references to HIS visits along with medical history files such as bulky media files related to lab tests or medical procedures. To maintain confidentiality of records, we do not keep all the records together at any other location, except that with the patient over which he has complete control. Although this mitigates the privacy issue, it makes the job of backing up the healthcard all the more difficult [12]. We propose a distributed query mechanism that keeps reference pointers to all these visit records at a central location which can be used to pull all the scattered EHRs required for restoration of the patient's complete medical account in case of damage to or theft of healthcard. All HISs may not share the same data model and hence the data dispersed may be in heterogeneous forms. To keep the records in a regular format, we use HL7 framework in which we translate EHRs retrieved from different sources using a common HL7v2 template. These HL7 based records encapsulated together represents the healthcard of the patient. Though the system is able to retrieve the PHR of an individual, keeping references to disseminated EHRs necessitates high security and availability. To provide high availability, we use Hadoop Distributed File System that allows replication of data while being highly fault tolerant and can be deployed on inexpensive hardware [13]. In this way, we are able to

provide a novel method to aggregate distributed health records that can be used as backup for the PHR of the patient.

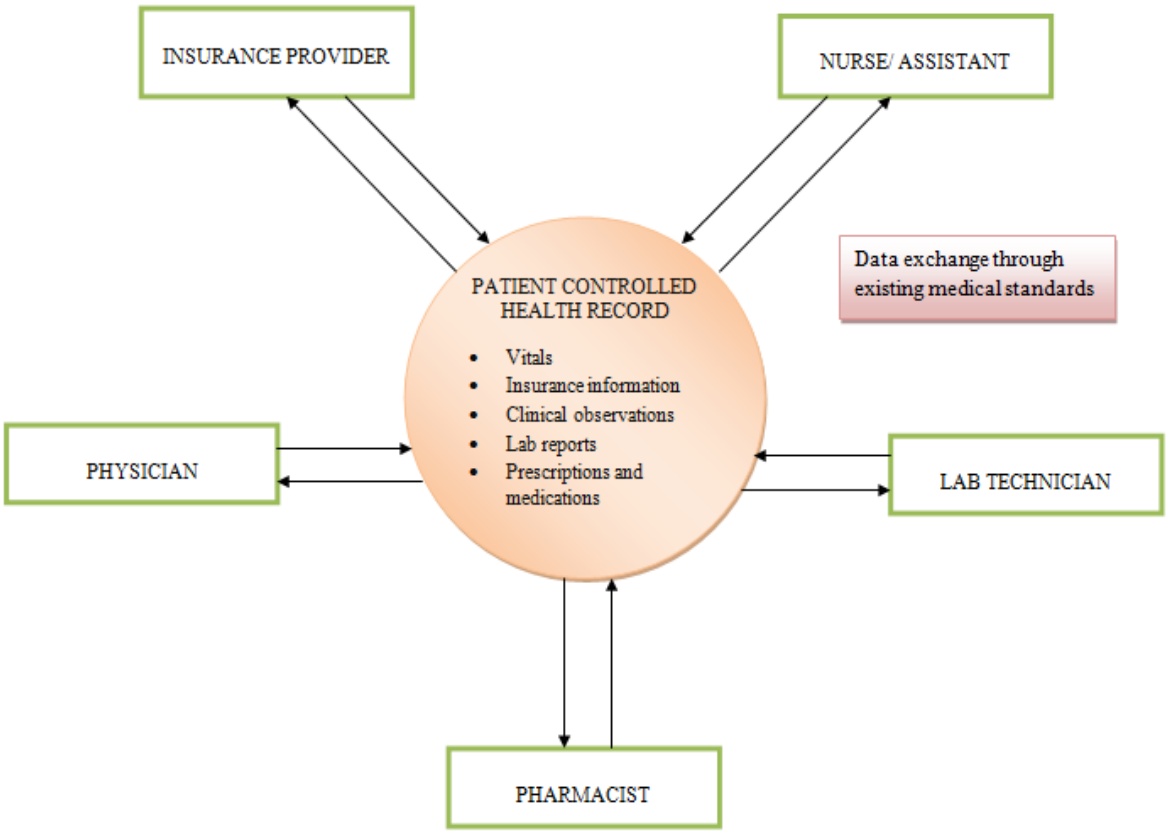

Figure 1 : Hybrid PHR model based on hub and spokes concept [7]

The thesis is divided into 8 chapters. Chapter 1 gives an introduction to the work done with clearly stating the motivation for the work, the problem and a brief about the solution.

Chapter 2 presents the Literature Review which includes little description work about related work done by people all over the world.

Chapter 3 gives insight into different PHR models, their usage and functionality, and a comparison among them.

Chapter 4 provides an introduction to Hospital Information Systems and describes three different open source systems used in the study.

Chapter 5 gives an overview to HL7 framework with explaining what exactly is an HL7 message, its components and how can they be used to transmit health information to achieve interoperability.

Chapter 6 explains what Hadoop Distributed File System (HDFS) is and different file operation are performed on it. The chapter also briefs about SequenceFiles and their use to solve the small file problem on HDFS.

Chapter 7 goes in detail about the design and implementation of the solution while also explaining about different components involved in the proposed system.

Chapter 8 states the result of the proposed system, with comparing HDFS performance with that of a local file system.

The Conclusion and a discussion about its future prospects are provided in the next section. The thesis is ended with listing the works referred in the research and Appendix in which Java programs to create SequenceFiles and read them from HDFS is provided along with steps to deploy GNU Health in local environment.

CHAPER 2: LITERATURE REVIEW

In the past years, people's interest in keeping track of their medical information has widened. They want access to their health records in a convenient way while without distressing about the confidentiality and security of the records [12, 14]. This has led to exploration of PHRs and their different models, research in the area of integration of medical information and development of medical standards for transfer of information and interoperability. Adoption of PHR is evident in various parts of the world [2, 5, 15]. Large organizations such as Google and Microsoft have also invested in personal health by providing online health repositories, namely Google Health and Microsoft HeathVault, though Google has discontinued Google Health as it didn't catch on [16]. A brief evaluation of user-end features of both the products can be found in [17].

A simple standalone PHR may not require to integrate with other sources, but to provide better healthcare services it is essential to interact and incorporate with multiple healthcare facilities. A lot of research has been done in the field of integration of medical information. A cloud based solution has been proposed by Ankur Agarwal, Borko Furht and Vivek Tyagi that allows integration of information from various health record systems using existing medical standards and make it available to a patient or a doctor through a web portal [18]. They propose a cloud architecture that keeps the patient records centrally which can be accessed anytime by an authorized user. The information kept on the cloud is the integrated information received from various HL7 compliant diverse sources like information systems, medical record systems and imaging systems.

In [19], Anant R Koppar and V Sridhar proposed a Tele-Health Medical Diagnostic System that has been deployed in few healthcare facilities in Karnataka, India. The system helps in monitoring patients remotely while also aiding in diagnostic process. It uses an integrated EHR comprising of patient related digital information from different visits that forms the backbone of the system. The EHR is also capable of exchanging data with local or regional hospital so as to provide continuity of care especially during medical referrals.

A clinical data exchange system using Ensemble Integration platform has been developed in China by Wang Yu, Guo Long, Tian Yu and Jing-Song Li that collects clinical data from various medical institutions and transforms and uploads it into a centralized data repository

(data center) using HL7 standard files [20]. The data center parses the HL7 files and store the information in a central schema to which authorized access is given to providers and patients with the help of ASP.NET technology.

A model for sharing patient generated healthcare data with healthcare provider's EHR system has been proposed by Walter Sujansky and Douglas Kunz that allows structured data retrieval using HL7v2 message combined with HL7 CDA (Clinical Document Architecture) through secure transfer using Direct project's email standards [21]. The proposed model collects the Patient Generated Health Information (PGHI) in a centralized repository that is able to communicate with different healthcare professionals with the help of Direct standard using query response mechanism.

In [22], Daniel James Baldock, Vadim Georgiovich Svinenko, Kerri E. Marshall and Karen Calhoun presented a hybrid architecture for sharing of health records that keeps the actual patient records at different locations but maintains patient identifying information at a central component that is to be used for retrieval of distributed records.

The Dutch EPD system also stores patient records into decentralized component known as GBZs managed by each hospital locally. To access the dispersed records, a reference index for each patient is maintained at a central component storing references to all the particular patient's records and the idea was extended by the author that involves elimination of centralized component to reduce privacy and security risks by explicit transfer of references, when required, under the control of the patient or the doctor [9, 23].

The work proposed in this paper provides a novel manner of storing integrated records on the healthcard that is highly available along with a backup on a digital vault in the cloud which can refurbish the details of the card through distributed query mechanism.

CHAPTER 3: PERSONAL HEALTH RECORD MODELS

In this chapter, we introduce different type of digital healthcare records available to either the patient or to the health facility. In later sections, different personal health records models like standalone, tethered and integrated are introduced and compared by considering their merits and demerits. In the end, we also expand on our hybrid PHR model that was introduced earlier.

3.1 Various Health Records

In today's healthcare sector, there has been a surge in the use of digital healthcare records instead of the paper base records due to easy management of patient data. Different users use various types of digital health records to realize their specific needs.

An *Electronic Medical Record* (EMR) is a digital version of paper based records held by a single clinic or any other solo healthcare organization. EMRs are restricted to only a single practice and they only create or manage medical data about different patients being treated in that particular facility. They allow a healthcare provider to keep track of medical history of a patient, although in limited manner. EMRs can also be used to identify patients and check on them based on certain parameters like whether a patient is vaccinated for a particular disease or not or if patient's blood pressure levels are at par or not [24]. Though they offer great benefits to patients but they do not communicate with other facilities, limiting its functionality. Thus, when a patient goes to some other healthcare practice for any reason, his data is not readily available and physicians are unable to monitor patient's health effectively.

An *Electronic Health Record* (EHR) does not only refer to a digitized health record but may encompass a whole electronic system that can automatically collect data about a patient or a population of patient. They may include different range of data like patient's medical history, his medication, vaccinations, allergies, lab results and even personal and demographical information. EHRs are more capable than EMRs as they allow sharing of data among different healthcare organizations providing a broader view of patient's current medical status [24]. They typically implement interoperability standards to allow easy sharing of data among diverse systems. It reduces data redundancy and provides more up to date records that aid providers in delivering better healthcare services. EHRs are generally under direct control

of healthcare facilities thus minimizing the authority of a patient on his own healthcare information.

A *Personal Health Record* (PHR), in simple terms, can be a person's paper based or computer file record that holds information like doctor visit details, medications, insurance information, lab reports, and much more about the patient. PHRs are different from EHRs and EMRs as they represent an electronic record that is shared, managed and controlled by the patient himself [25]. It provides a longitudinal view of significant health information via a single interface. There is no set standard for what a PHR may include or what functions it might provide though a PHR must aid an individual in the process of self care while also additionally facilitating communication with healthcare professionals to improve the quality of healthcare delivery process. To have such characteristics, a PHR must include:

- Identification information – patient's name, address, telephone, demographics
- Diagnosis information – patient's current and past diagnoses
- Lab Reports – lab tests ordered and their results along with standard medical images such as X-Ray reports, ultrasound, mammogram etc.
- Medication – history of patient's medication along with his current medicine information
- Physical records - Patient's physiological parameters like weight, height, temperature, glucose level etc. measured over time
- Allergy record – list of patient's allergies identified by providers
- Immunization record – list of immunization given to the patient during his lifetime
- Insurance information – patient's insurance plan data along with deductible and non-deductible charges
- Provider comments – comments noted by various healthcare providers like physician, pharmacist or nurse

Based on various attributes like type of data being stored, sources of such data and different use cases of the system, PHRs can be seen as anything between a standalone PHR that keeps all the medical information together on a patient's device over which a person has complete control and an integrated PHR that keeps the complete data in a centralized repository that can be accessed over the web by the patients and providers with proper authentication an authorization [2]. An integrated PHR generally pulls information from various hospital

information systems and then integrate it together but if the PHR is modelled to work with only a single Hospital or healthcare facility, it is said to be tethered to that particular organization. A difference among the three is seen in Table I.

3.2 Standalone PHR Model

Standalone PHRs are generally individually maintained or the privilege is extended to certain individuals like close family members or family doctor. Such type of PHR can be kept by the patient in a secure device ensuring portability of the records. They generally constitute of patient generated health information such as blood glucose measurement, weight measurement or other vital data captured by body sensors connected in a Body Area Network (BAN). These types of PHR offers many benefits like accepting input from patient, managing diet charts, and receiving medication reminders. Although patient generated data can be really helpful but they might be trivial if they are not mapped on to well defined parameters so that the data may help a clinician in the diagnostic process. While providing confidentiality and safety of records, standalone PHRs cannot connect to various information sources making them islands of information not able to contribute much [3]. Also, reliability of patient measured data is always dubious. For example, a patient may be able to record correct weight or temperature measurements but he might not be able to record accurate laboratory values such as haemoglobin or cholesterol levels needed by a physician [6]. Standalone PHRs are easy to implement and use as all the data is consolidated into a single destination kept under the direct control of the patient or related affiliates. These destinations can be anything like smart cards, USB, mobile phones or any other manageable device providing both portability and confidentiality of records. Figure 2 and Figure 3 measures the complexity and functionality of standalone PHRs against other PHR models.

3.3 Integrated PHR Model

In integrated PHRs, information about the patient is collected from various sources such as imaging systems, pharmacies, EHRs kept in hospitals or clinics and even patient entered data, although for selected fields [5]. It provides a complete view of relevant health information for both patient and the healthcare provider and are equipped with much more robust backup systems [2]. Integrated PHRs not only allows access to the healthcare facility kept medical records but also serve as communication channel between patient and healthcare providers.

Such PHR helps in reducing clinical errors and also in elimination of duplication of data. They are generally made available over the web through some sophisticated interface with the use of proper authentication and authorization services for secure access but confidentiality of records may still be compromised as the patient does not have complete control over the records. Such integrated systems may need to take extra measures for providing privacy and security of records to the patient.

Integrated PHRs can either be push based system or pull based system [12]. In push based system, the central source receives data from different clinical sources with the authorization of the patient. In pull based system, the central repository retrieves data from all the sources in which a patient's information is stored. Such pull based systems need to keep track of all the primary sources holding patient's records and require proper permission from both patient and the healthcare facility for accessing those records.

Although integrated PHRs offer great benefits to health concerned individuals, such systems are difficult to realize in practice. This is due to requirement of integration of medical information from heterogeneous sources. To achieve interoperability, we need well established global medical standards that can be easily implemented by various information systems in question. In recent times, many standards have come to light such as OpenEHR, HL7 and CCR (Continuity of care records) but only few of them have been globally accepted and are currently in practice. Also, competing facilities do not want to share information with each other and there is lack of trust among them, hence causing hindrance in the path of achieving the ideal integrated PHR.

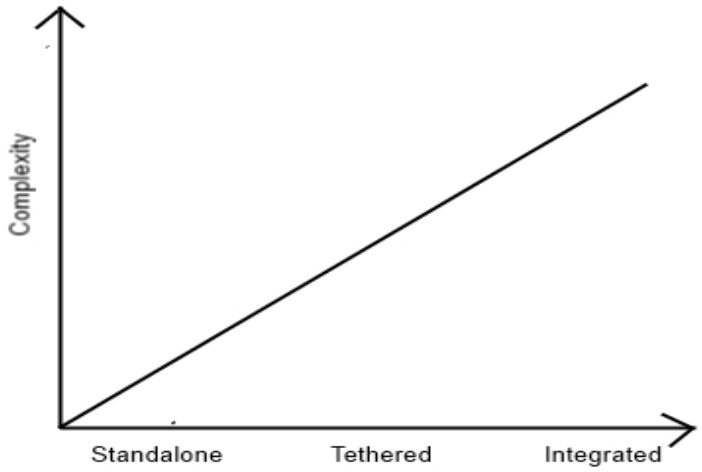

Figure 2: Complexity of various PHR models

3.4 Tethered PHR Model

Tethered PHRs can be seen as simplified version of integrated PHRs in which a person's health record is tethered to a particular medical plan or to a single healthcare organization which may offer complete or partial access to health records through web portals. It can provide additional functionality to a user like ability to email medical providers, renew medical prescriptions and even make future appointments with a physician [5]. Even though they offer many advantages and are also relatively simple to implement but when the patient switches to another medical plan or facility, his data may not be conveyable due to system incompatibilities or due to unwillingness of associations to share data.

Table I Review of various PHR models [12]

Attribute	Standalone PHR	Tethered PHR	Integrated PHR
Complexity	Low (but backup complex)	Moderate	High
Access	Web based, memory sticks or card readers	Web based	Web based
Confidentiality	High (Complete control by the patient)	Low (tethered facility is in control)	Low (can be improved by letting patient set access controls)
Security	High	Low (depends on the facility's security system)	Low (can be improved by installing a third party central security system)
Information sources	Patient entered data, body sensors	Facility's EHR, physician comments	Patient entered data, body sensors, Facility's EHR, labs, pharmacies, doctor comments
Challenges	Loss or theft of device, non standard data input from the patient	Access to records is restricted and patient may not recover his records when he switches to another medical plan or facility	Use of common medical standards for interoperability and cooperation, level of trust among various affiliates

It can be seen that each PHR model has its own set of merits and demerits. To combine the advantages of integrated and standalone PHRs, we have proposed the use of hybrid model that will remain, at all times, in control of the patient so as to provide confidentiality of the records and also it can be used to communicate medical information to associated healthcare practitioners with the use of widely implemented medical standards such as HL7.

CHAPTER 4: HOSPITAL INFORMATION SYSTEMS

A hospital or a healthcare information system (HIS) can be seen as automated system that not only provides electronic storage of patient health records but also satiates administrative and financial needs of a healthcare facility. Usage of such systems not only improves quality of care provided to patients but also make healthcare services more affordable and easily accessible [25, 26]. A HIS may comprise of a data model that stores the patient record in a structured format electronically, authorization and security mechanisms for protection of patient data, information communication tools for promoting interoperability, data analysis and visualization tools for enhancing care services, and a for making the system simple to operate. In the recent years, there has been an increase in the usage of network based application software as care delivery systems due to their easy availability and deployment combined with a smooth learning curve, although such factors vary for different developers or vendors. Some popular HIS applications are OpenMRS, FreeMED, GNU Health, OpenEMR, CottageMed, IndivoX etc. We describe the three most popular among them in the next sections.

4.1 OpenMRS

OpenMRS is an open source medical record system that was created in 2004 through a non-profit collaboration led by Regenstrief Institute and Partners in Health, used readily in developing countries [27]. OpenMRS provides a platform to create customized medical record system through their modular design and use of open source components. The core application is a client-server application having a multi-tiered architecture. OpenMRS's data model closely resembles Regenstrief data model uses a concept dictionary that defines different medical terms as concepts. The dictionary has concepts defined for medical procedures, lab tests and results, diseases and symptoms, and even medications [28]. It even has mappings to international standards such as SNOMED CT, LOINC, ICD-10 etc. The base application is built in Java and the user interaction is provided through a web interface built using JavaScript and JSPs. OpenMRS is able to collaborate with other networks and EMR application with the help of HL7 interoperability standard [29].

As stated in [27], "*OpenMRS provides a software platform and a reference application that enables design of a customized medical records system. It is a client-server application,*

which means it is designed to work in an environment where many client computers access the same information on a server".

There are several layers to the system:

- The data model is inspired from the well known Regenstrief data model.
- The API providing interfaces to the access the data model.
- The web application including web front-end and modules that extend the core functions.

The core OpenMRS application comprises a web application, programmed in Java and JavaScript and a number of open-source component applications, maintained by other open source communities, including MySQL, Apache Tomcat and Hibernate. The architecture based on MVC model, as shown in Figure 3 comprises of three layers:

- Presentation Layer - The presentation layer includes a sophisticated web interface built using JSP and JavaScript. JQuery is used to simplify the interactions with JavaScript and the browser.
- Service Layer - The service layer is responsible for managing the business logic of the application. It uses the Spring framework for managing transactions between service layer classes and thus providing functions like authentication and logging.
- Data access Layer - The data access layer is an abstraction layer representing the actual data model. It uses Hibernate framework as the Object Relational Mapping tool which manages relational database changes in a database independent way.

OpenMRS is fulfilling its potential as a low cost, rapid development, open source application for developing patient treatment and management systems in resource-poor settings. The generality of the core application design-based on the clinical encounter with flexible addition of observations linked to concepts will likely support extension to information management in a number of primary healthcare settings in developing countries [30].

Some of the most important features that OpenMRS provide are:

- A central medical dictionary that contains all the definitions for medical data so as to provide interoperability among different systems using coded data
- Security using authentication and role based access
- Storage and maintenance of patient's data involving lab tests, demographics, visit and emergency data

- HL7 standard support with FHIR support in development
- Addition of modules to extend the functionality of the basic system
- Support for different languages and localizations

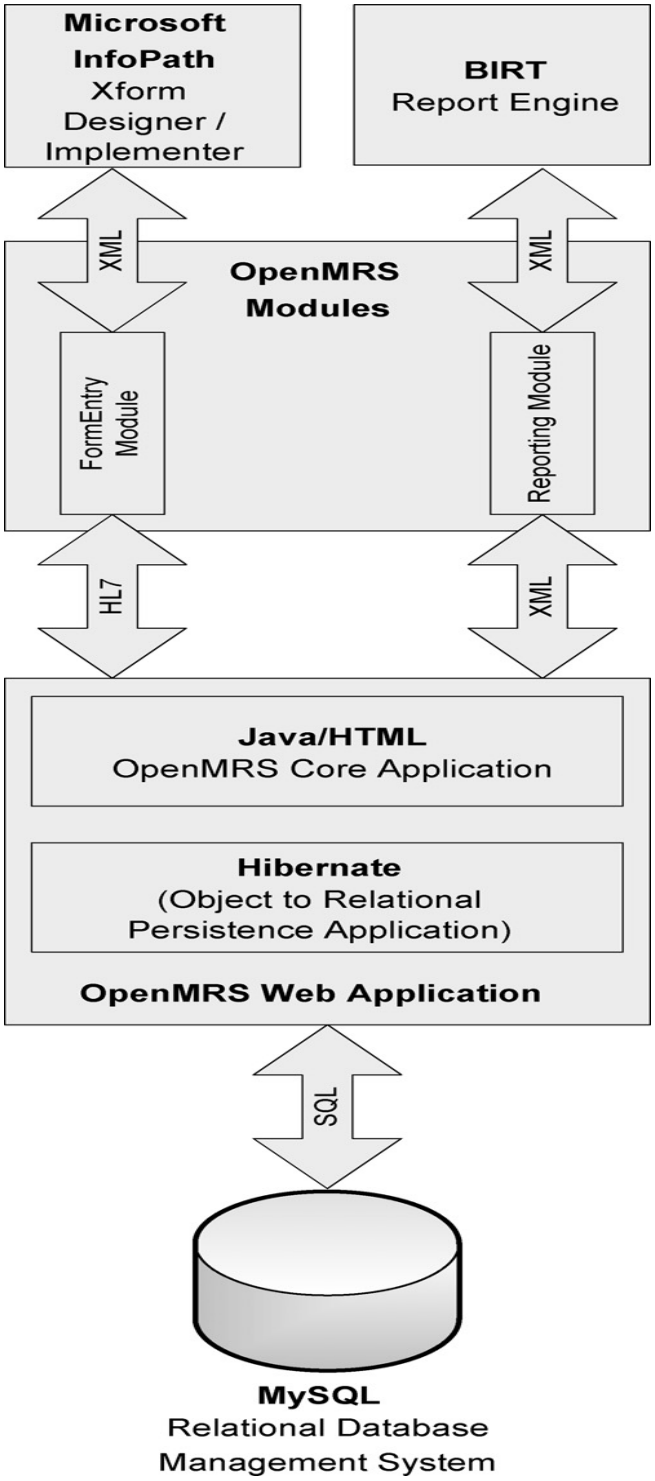

Figure 3 : OpenMRS Technical Architecture [30]

4.1.1 Data Transfer using REST web service

Communication between OpenMRS (HIS) with an external system occurs with the help of REST web service. REST (Representational State Transfer) defines a set of architectural principles used to design Web services that focus on system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages. OpenMRS provides REST web service through a module that can be used to access remote resources. These resources are returned as JSON objects that can be then transferred over the network via the REST API.

A REST service includes methods to oversee how resource states are addressed and transferred over HTTP [31]. A concrete implementation of a REST Web service follows four basic design principles:

- Use HTTP methods explicitly
- Be stateless
- Expose directory structure - like URIs
- Transfer XML or JavaScript Object Notation (JSON)

The basic REST design principle establishes a one-to-one mapping between creates, read, update, and delete (CRUD) operations and HTTP methods:

- POST method is used to create a resource on the server.
- GET method is used to retrieve a resource from the server.
- PUT method is used to change or update the state of the resource.
- DELETE method is used to delete a resource from the server.

4.1.2 OpenMRS Medical Dictionary

OpenMRS uses a concept dictionary that defines all the medical terms such as observations, diseases, symptoms and medications in terms of coded data [29]. This coded data can then be used for exchanging data over the network without losing its semantics, thus facilitating interoperability. Concepts are seen as individual data points collected for a patient. The dictionary not only assigns codes to various clinical terms but also provides meaning and description of each term, relationships with other concepts, classes and attributes of different concepts, and also mappings to other popular medical dictionaries such as SNOMED and LOINC.

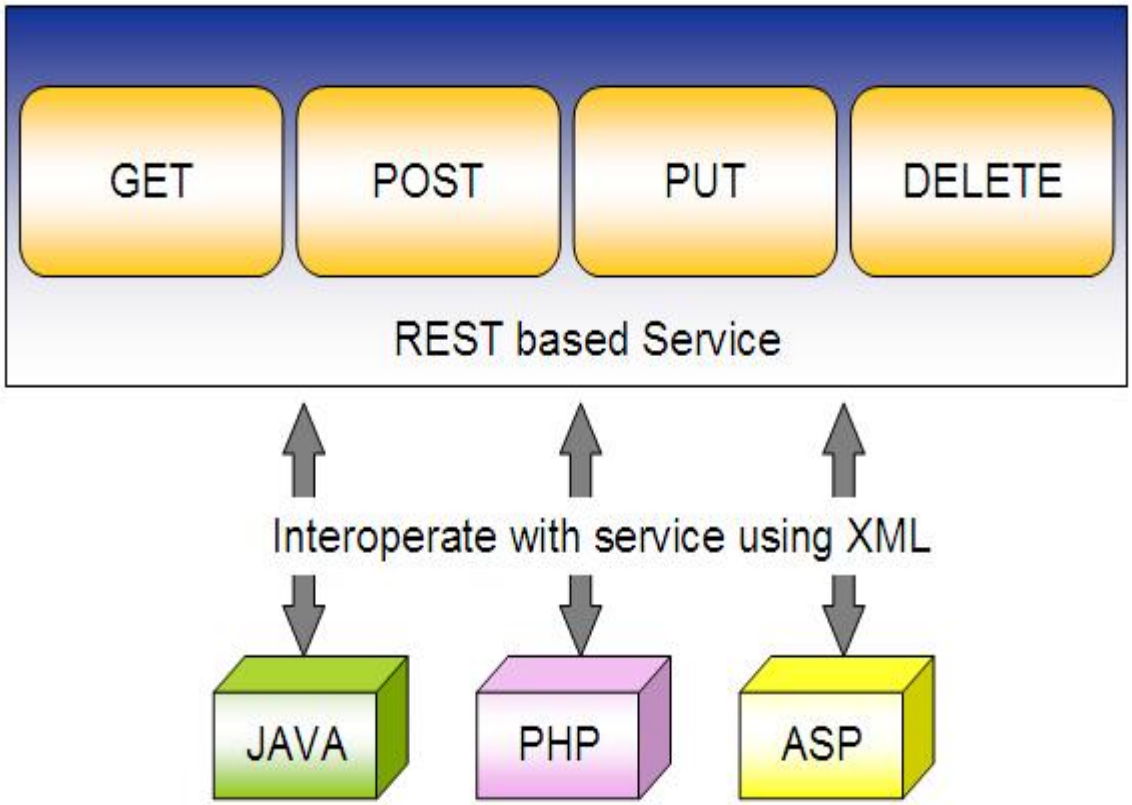

Figure 4 : REST Web Service Methods [32]

A concept can either be a drug, a medical test, coded result, a symptom or a medical problem. For example, *Haemoglobin value* is a medical test that has a numeric code defined as 21. It is stated by the dictionary that this concept takes a numeric value as answer with measuring units as g/dl and in the range of 12.0 to 16.0 g/dl. A snapshot of the concept as defined in the dictionary is given in Figure 5.

The latest versions of OpenMRS has moved to a more robust MVP CIEL dictionary created through joint contribution from Millennium Villages Project (MVP) and Columbia International eHealth Laboratory (CIEL) and has mappings to well known international interoperability standards such as SNOMED, ICD-10, LOINC etc. Efforts are also being made to translate the dictionary into different languages [33].

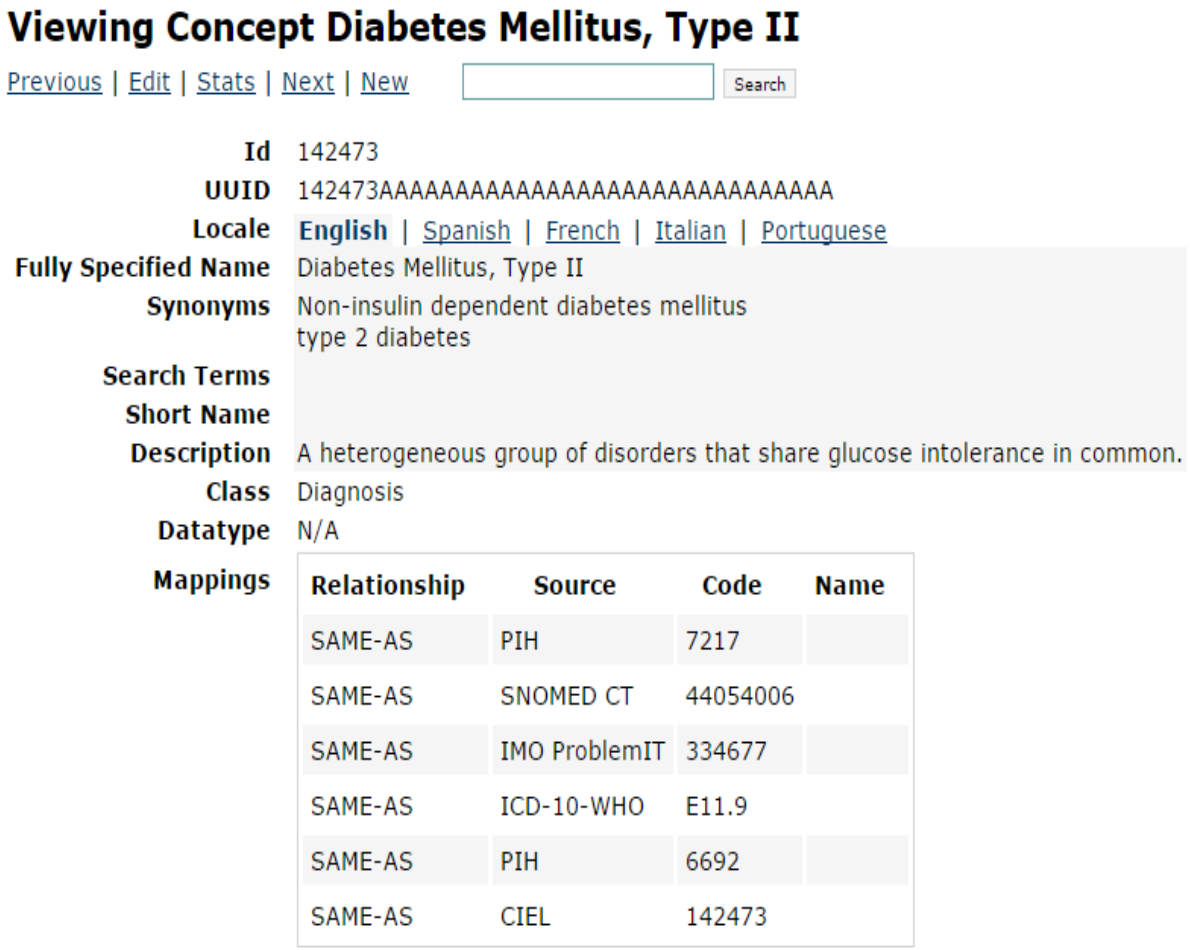

Figure 5 : OpenMRS Concept Dictionary mappings example

4.2 OpenEMR

OpenEMR is another open source health management system that is able to run on multiple platforms. It is ONC certified and is working towards Meaningful Stage II certification in the United States [34]. The application is a web based application that can be accessed through a browser and is built using PHP, Apache Http Server and MySQL database. Through OpenEMR, a healthcare professional is able to track patient demographics, schedule appointments, store visit based records, manage prescriptions and provide billing services. It allows patient documents to be encrypted and provides permissions based per-user access control for added security. It also allows import of standardized code sets like ICD-10 and SNOMED. The typical deployment architecture for OpenEMR is given in Figure 6 that shows what components are involved and where those components are placed.

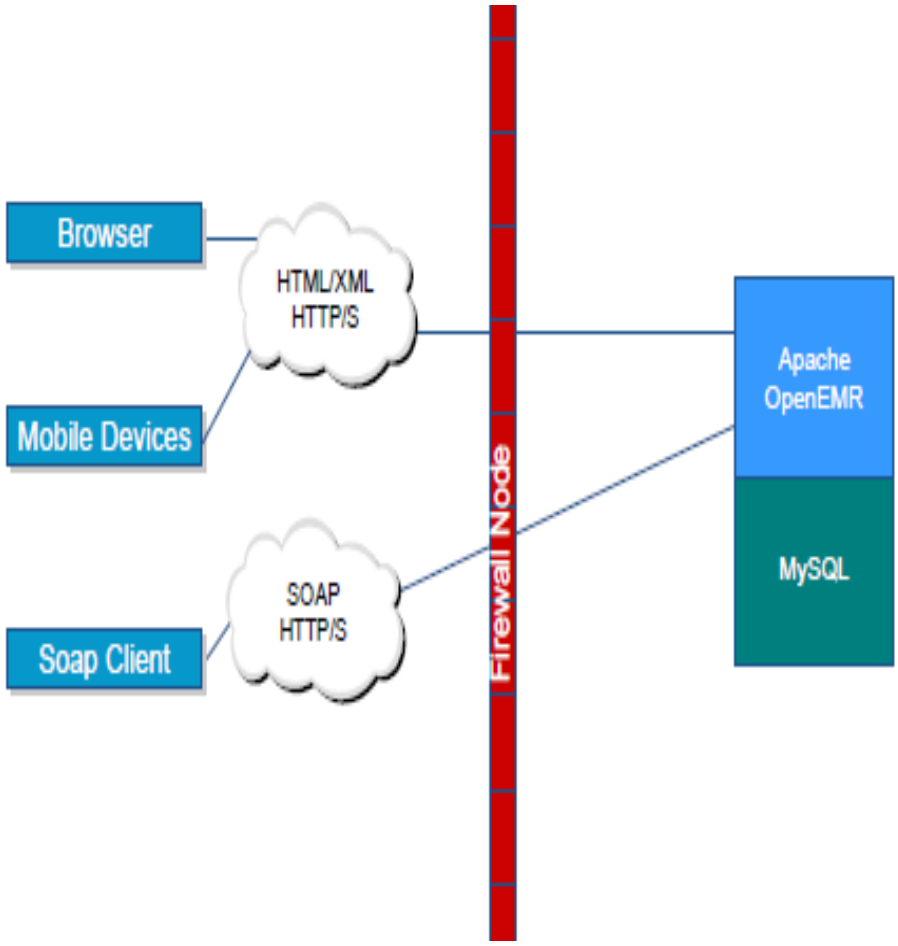

Figure 6 : OpenEMR typical deployment architecture

4.2.1 Features of OpenEMR

Some important features of OpenEMR are [35]:

- Free and ONC certified – OpenEMR is open source, free to use and is also ONC certified
- Personalized patient demographics – It can store variety of demographics data like name, date of birth, gender, marital status, contact information, primary provider, insurance data etc. and can be customized according to facility's needs.
- Patient appointments administration – patients can set appointments at different but connected facilities and can get notifications over mail or sms.
- EHR (Electronic Health Records) – OpenEMR stores medical information about a patient as EHRs combining various sections like encounter data, medical problems, vital stats, immunization data ,lab results and doctor comments

- Medication and prescriptions – The application can search for drugs name online and track patient's current and past medications and also create prescriptions while also providing reminders to patients.
- Billing – It can use coding systems like ICD9, ICD10 and SNOMED and can, thus, create fully legible medical bills with also incorporating medical claims
- Security – OpenEMR provides security of records by encrypting them through an external module while also providing authentication and role base access controls.
- Report generation – The software also generates reports involving patient lists, immunizations, sales, insurance eligibility etc.
- Multilanguage support – OpenEMR provides Multilanguage support within the same clinic and also other language support can be added easily.

Although OpenEMR offers many benefits, it still lacks a good interface and it provides no external interface that can transmit medical data to a remote system, unlike OpenMRS and GNU Health.

4.3 GNU Health

GNU Health [36] is also free and open source health information system that started as a project for Primary Care facilities in developing countries but is now used by both public and private institutions in different parts of the world. The system is written in Python and uses the Tryton general purpose application platform. GNU health allows both patient management that includes managing patient history, demographics, encounters, medications etc. and health center management overseeing finances, labs, pharmacies and even human resources. It can be used at different scales like in single doctor clinic or in public hospitals by using centralized or distributed type of deployment [37]. GNU Health is relatively new and although it provides a FHIR (Fast Healthcare Interoperability Resources) server for exchanging data over HTTP, resources that can be retrieved are limited with no write functionality [38, 39]. Figure 7 outlines the modular design of GNU Health based on Tryton framework. Such modular design of the software assists facilities to build over the core component so as to provide department specific and localized support. It also allows one to use GNU Health in different scenarios like being used as EHR systems in clinics or being used as complete hospital management systems.

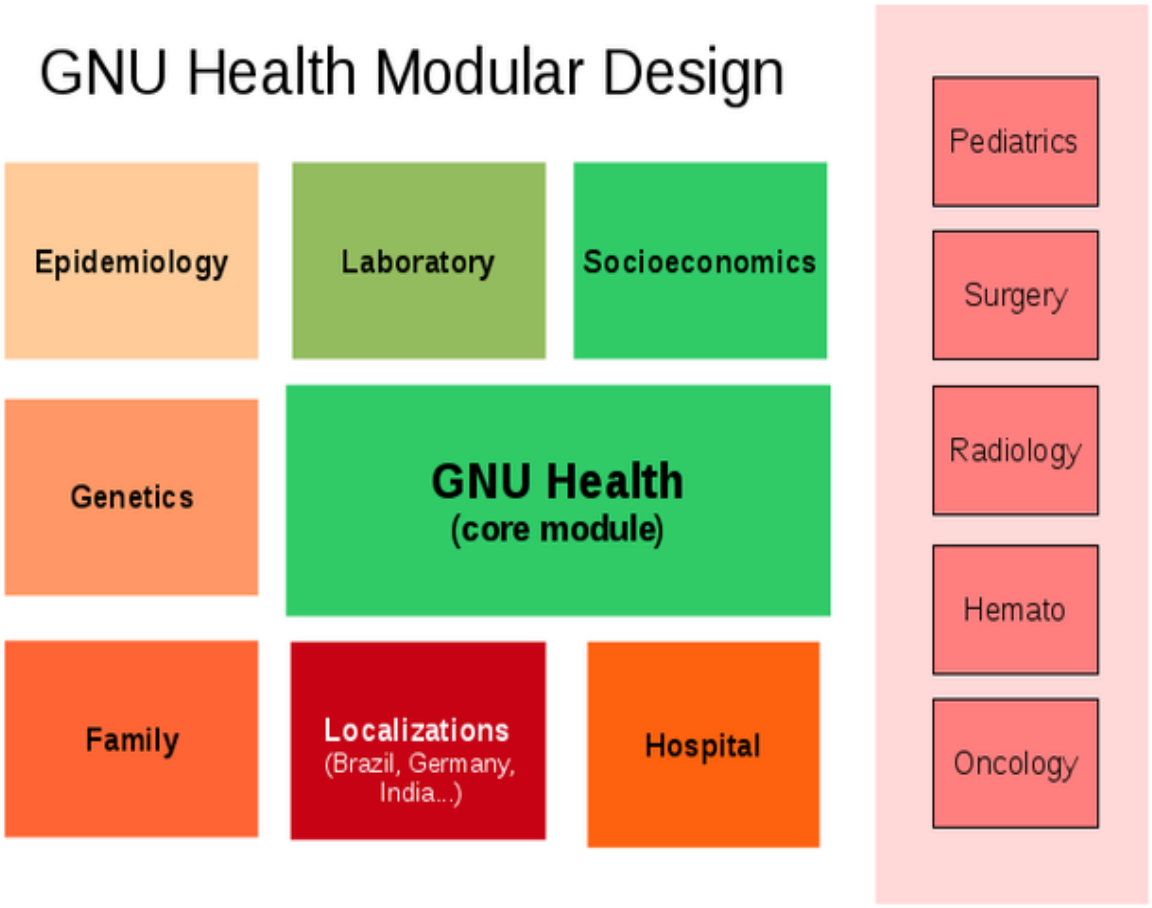

Figure 7 : Modular Structure of GNU Health [37]

The primary domains that GNU Health includes are [37]:

- Individual and community management – It includes managing patient demographics, domiciliary units and other operational areas.
- Patient management – It includes management of patient records like lab reports, encounters, hospitalizations, medical history, lifestyle, Socioeconomics etc.
- Health center management - It includes management of finances, stocks, pharmacy , laboratory, beds, operating rooms, appointments, supply chain management, human resource management for the facility
- Information management - It involves reporting, demographics analysis and cause and effect analysis for health and disease conditions (Epidemiology)

4.3.1 GNU deployment scenarios

GNU Health is scalable in functionality, database size and transactional volume. For instance, one can install GNU Health in a single doctor office, or in country public hospitals network. Depending upon the type of deployment, one will think about centralized (single instance) versus a distributed installation.

- Single GNU Health Instance: All the information resides in a single database, and it will be accessed via network from different workstations from the same health center (local area network) or from different health centers.
- Distributed GNU Health instances: Under this scenario, each health center has its own database instance, and information can be synchronized among health centers. This would be the case when you want to deploy GNU Health in a network of hospitals, where the communication infrastructure is suboptimal.

Choosing the deployment method requires careful study of resources like hardware available, network architecture, human resources, security and access control, and backup and disaster recovery policies.

4.3.2 Data transfer through FHIR REST Server

Fast Healthcare Interoperability Resources (FHIR) is an HL7 standard framework used to transfer resources in a flexible manner. It combines the best parts about HL7v2, HLv3 and CDA while focussing more on implementation [38]. GNU Health allows for remote transfer of patient information as FHIR resources sent through a REST server. FHIR standard is relatively new and is being highly endorsed be HL7 organization itself. GNU Health's FHIR REST server is still under development and as of now it only provides read access to the patient resources. Currently, it supports the retrieval of 12 resources [40]:

- Conformance: Describes the server's FHIR capabilities.
- Patient: Patient information, like email, address, SSN, etc.
- DiagnosticReport: Completed lab tests, but not the data
- Observation: Lab data, like Uric Acid values
- Practitioner: Health professionals and their information
- Procedure: Surgeries/operations

- Condition: Diseases/diagnoses
- FamilyHistory: Family histories of patients
- Medication: Medications (not prescriptions)
- MedicationStatement: Medications taken by a patient
- Immunization: Immunizations
- Organization: Institutions, departments, companies, etc.

Table II Comparison between OpenMRS, OpenEMR and GNU Health

Feature	**OpenMRS**	**OpenEMR**	**GNU Health**
Technology	Java	PHP	Python
Web or Desktop	Web based	Web based	Desktop
Open Source	Yes	Yes	Yes
Multi-platform	Yes	Yes	Yes
Online Support	Wiki + Forums	Forums + Mailing list	Mailing list
Website	http://openmrs.org/	http://www.open-emr.org/	http://health.gnu.org/
External Interface	Yes(send & receive)*	No	Yes (send)**
HL7 Support	Yes (receive)	No	Yes (FHIR send)
Vastly Implemented in	Developing countries (NGO support)	Primarily US based	Latin America
Pros	+Technical platform, can be easily extended or modified according to ones need +Large development community with active GSOC participation	+Very secure as can be accessed remotely through UI +Additional Medical features like mailing reports and scheduling appointments through downloadable modules	+Built in python using Tryton framework making it easy to extend by adding modules

Cons	- Medical knowledge required to add data through medical concepts	- No external interface to fetch data in background - Dated UI	- Comparatively new and has lot less support

*** REST interface that can receive HL7 messages for adding new patient or adding observations.**
**** FHIR REST server that can send medical information in JSON or XML format. (Limited as still in development)**

CHAPTER 5: HL7 FRAMEWORK

In this chapter, we give a brief introduction to different healthcare interoperability standards that are available today. The chapter also provides an overview of HL7v2 messages followed by examples of how ADT and ORU messages can be used to represent and transfer medical data. Later, we describe the evolving FHIR medical standard and finish with introducing Mirth Connect and over what protocols it can be used.

5.1 Interoperability Standards in Healthcare

Different HISs store information in diverse formats such as in relational database, structured documents like XML or JSON, or even digitized copy of physical records making it very difficult to exchange information among various facilities [41]. This leads to requirement of standard ways for exchanging data to make systems interoperable. Some of such standards are HL7, openEHR, Continuity of Care Records (CCR) and Continuity of Care Documents (CCD).

OpenEHR is a medical standard that defines a set of specifications delineating how medical data can be stored, managed and exchanged with various systems while its primary focus being to develop interoperable EHR systems from scratch [42]. The specifications define a reference model known as archetypes using which healthcare systems can be designed. The common archetype based design ensures interoperability among the built systems but it does not do much work with pre-existing systems. Archetypes are considered fairly complex and require relatively more resources for implementation. Hence, OpenEHR is used in large scale systems focussing on medical data sharing.

Continuity of Care Record (CCR) is an ASTM standard used to exchange clinical data. It is like a snapshot of relevant medical information or specific condition. CCR could contain information about the patient's current medications, allergies, recent visits, diagnoses, suggestions etc. from previous providers so that the next provider can deliver more continuous healthcare services. CCR standard is designed to allow for information to be pulled from one or more sources and organized into a single XML file. It has sections like:

- CCR identifier (identifies referring and referred clinician)
- Patient identifying info

- Financial/ Insurance data
- Patient health status (vital signs, diagnoses, allergies, current medication, lab results)

CCR is not a historical document containing physician notes, transfer information or a discharge summary. Most importantly, the CCR standard is neither a substitute nor a replacement for an EHR [43]. Healthcare industry seems to be moving away from CCR as it is generally only used to relay patient information from one clinician to another so that other clinician knows the patient current status.

HL7 version 2 (HL7v2) is an ANSI certified set of international standards used for medical data exchange. HL7v2 medical standard is the most popular interoperability standard in use [44]. It provides a framework for sharing and integration of medical information. The standards dictate how information should be structured and transferred among similar or dissimilar systems. It is based on the assumption that health data is exchanged among various applications due to occurrence of some event such as patient admit or discharge, lab tests ordered or lab tests completed, pharmacy order or some medical procedure scheduled. HL7v2 messages are basic string message in which data is segregated with special symbols with each field flexible enough to hold different type of data. Due to this adaptable nature HL7v2 is the most implemented interoperability standard in the medical industry.

HL7 version 3 (HL7v3) is the next stage for HL7v2 introduced by the same organization in 2005 bringing formalism to the standards. It uses an object oriented Reference Information Model (RIM) to attach semantics to each part of the standard. RIM defines a generic set of classes which are used to derive more specific concepts. This makes the standard more consistent but less customizable. In HL7v3, medical information is encoded as XML messages defining complex structure rules as compared to simple structured HL7v2 messages. It uses an XML Schema Definition (XSD) to define the message type and describe various components of the message. It has not been as popular as HL7v2 due to stringent modelling rules, being difficult to realize.

Clinical Document Architecture (CDA) was introduced by HL7 as a part of family of HL7v3 standards that defines a mark-up standard for specifying the structure and semantics of clinical documents that can be exchanged between patient and providers [45]. CDA documents can be transferred independently or can be transferred into HL7v2 or HL7v3 message. A CDA document is a defined and complete information object that can include

text, images, sounds, and other multimedia content. The document can be sent inside an HL7 message and can exist independently, outside a transferring message [46]. It is a part of HL7 v3 family of standards but it is a complex standard that can be challenging to implement. It also used RIM and due to this underlying information model in CDA and HL7v3, it makes them difficult to comprehend and use and lack of it in HL7v2 makes it flexible and easy to implement.

Continuity of Care Document (CCD) is built using CDA elements and contains data defined in CCR. It was formed by collaboration between ASTM and HL7. It can be seen as a restricted version or a use case of CDA. CCD defines certain templates that are to be filled in the document. These templates can be medication, allergies, immunization, family history, lab results etc. The data contained in each of the templates is set by CCR but CCD is quite complex than CCR. [47]. We can use CCD for storing all the data defined by the templates and then the document can be sent as HL7v3 message or it can be embedded into a HL7v2 message into certain segment of the message but majority of that information would be redundant as visits observation including vital signs, medication, lab results etc. can be sent directly through HL7v2 message.

Documents like CCR, CDA and CCD hold static information, i.e., a snapshot at a particular time. They cannot incorporate new information or any changes unlike standard HL7 messages. [48]. Each time new information is added, we will need to prepare a new document that will represent patient information for that particular time and store it with previously retained documents which will not be efficient on storage and processing, though, it can be useful for transferring images and special medical documents as those can be encoded into base64 CCD document and then embedded into a message itself providing support for transmission of various types of data [21].

5.2 Overview of HL7

Each HL7 message is made up segments and fields. A segment identifies what kind of information does a message contains like patient identification, diagnosis, insurance etc. whereas the fields contains the actual information like patient name, address, insurance id etc. Each segment is separated by carriage return symbol and each field is separated by pipe symbol. Every message contains a required MSH (Message header) segment that identifies

sender and receiver and also the type of message. More details are given in the below sections.

5.2.1 Components of a HL7 message

An HL7 message contains one or more *segments*, each of which identifies the type of data that the message contains like diagnosis, insurance, demographics, or observations. A carriage return character ('\r' - 0D in hexadecimal) is used to separate one segment from another and hence each segment is displayed on a different line of text.

Each segment consists of one or more composites, also known as *fields* that carry the actual information being conveyed such as person's name, observation value, medicine dosage, insurance id etc. A pipe ('|') character is used to separate one composite from another and if a field contains other composites, these sub-composites (or sub-fields) are normally separated by hat character ('^'). If a sub-composite also contains composites, these sub-sub-composites are normally separated by ampersand character ('&').

5.2.2 Sample HL7 message

Consider the following ADT^A01 sample message having various segments such as MSH, EVN, PID, PV1 etc. Some common segments are:

- MSH (Message Header) is always the first segment and marks the starting of new HL7 message. It carries information like source and destination of the message, date of the message, type of message, character set of the message and much more.
- EVN (Event type) holds information about the event triggered that required the sending of the message to receiving application.
- PID (Patient identification) segment is used to transmit information that can be used to identify a particular patient and also other personal information about the patient such as a unique patient id along with patient's name, date of birth, gender, race, address, contact information, citizenship and even death indicator.
- PV1 (Patient Visit) segment is used to communicate visit specific data such as attending doctor, referring doctor, visit type etc.

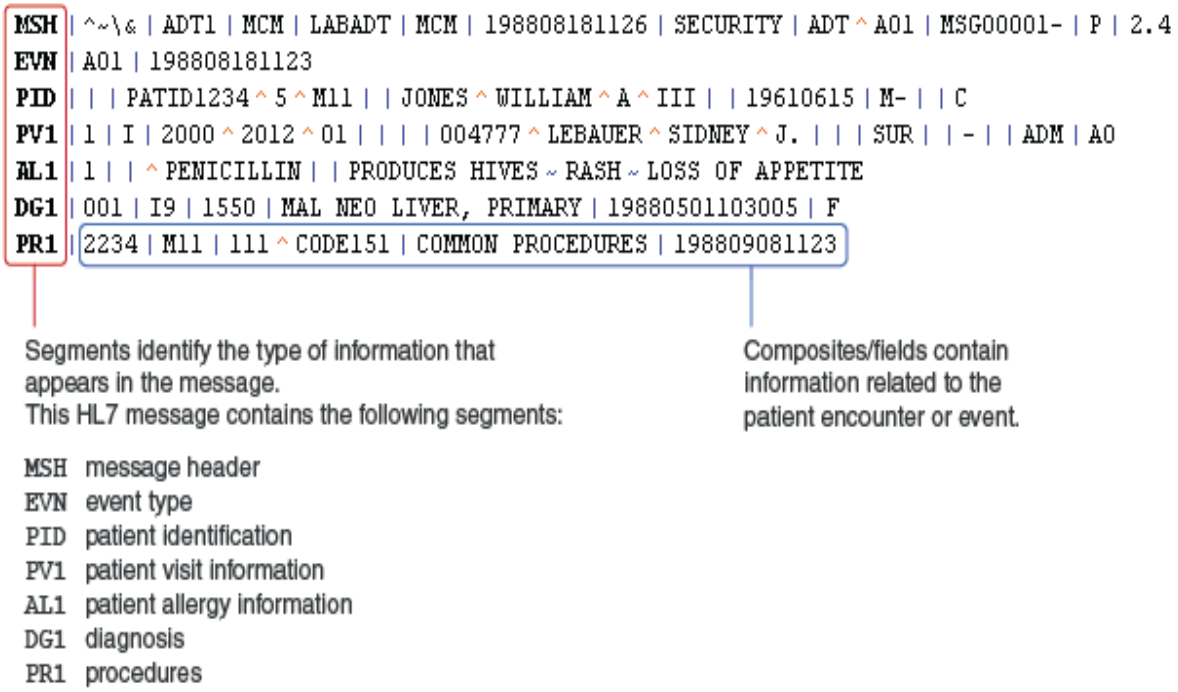

Figure 8 : HL7 Message Example [49]

5.2.3 Categories of HL7 message

There are four primary HL7 standard message types**:**

- Patient Administration (ADT)
- Orders (ORM)
- Results (ORU)
- Charges (DFT)

HL7 ADT messages are used to carry patient demographic data for HL7 exchanges and they also supply crucial information about trigger events such as whether patient got admitted into the facility or whether he got discharged, if the patient has transferred from some other healthcare centre or in case of new patient registration. Some of the most significant segments in the ADT message are the PID (Patient Identification) segment, the PV1 (Patient Visit) segment, and seldom the IN1 (Insurance) segment that is used to transmit insurance related data like insurance provider, policy type and its start and end date. ADT messages are universal in HL7 processing and are among the most extensively used among all message types.

According to [50], "*The HL7 ORU message is used to transmit observations and results from the producing system/filler (i.e. LIS, EKG system) to the ordering system/placer (i.e. HIS, physician office application). It may also be used to transmit result data from the producing system to a medical record archival system or to another system not part of the original order process. ORU messages are also sometimes used to register or link to clinical trials, or for medical reporting purposes for drugs and devices.*"

Types of observations reported in the ORU message typically include:

- Clinical lab results
- Imaging study reports
- EKG pulmonary function study results
- Patient condition or other data (i.e. vital signs, symptoms, allergies, notes, etc.)

ORU message can be seen as a structured report in which each observation is divided into individual articles, and then they are separated into fields/composites. Generally, ORU messages are not used to carry images but they can carry reference pointers to large audio/video files which then can be accessed remotely. Apart from pointers, they can use varying data types like numeric, normal text or coded data. An ORU message is composed of the OBR (Observation request) and OBX (Observation) that offers the following functionality:

- The OBR (order) segment is typically used as the header of the report or a section of the report. It consists of vital information about any order/ lab test that is to be filled. This may include fields like order number, request date and time, observation date and time, ordering provider and additional notes and comments. This segment is part of a group that can be used more than once for each observation result that is reported in the message.

- The OBX segment (filler) is used to transmit actual observational data that was ordered in the OBR segment. The fields included in this segment carry information like observation name (RBC, glucose level etc.), observation value, date and time of the observation and information about the responsible observer that can be used for auditing. As mentioned, observation value can be can be textual, numerical or of coded type.

5.3 ADT - A28 HL7 Message

An ADT event A28 message is used to add person's information into the receiving facility so that the sender and receiver have the same information about the individual in question. It contains various segments like PID that is used for patient identification and carries personal information like name, date of birth and gender. It also has a NK1 segment that can carry next of kin information for emergency situations. It can also carry diagnosis, allergy and insurance information as in Figure 9.

```
MSH|^~\&|OPENMRS|ABC CLINIC|JAVA_APP|QUERY_SYSTEM|20150409000000|HUP|ADT^A28^ADT_A05|ADD PERSON INFO|P|2.5|1|||AL||ASCII
EVN|A28|20150409000000|||1
PID|||2222^^^PATIENT IDENTIFIER||Patient^XYZ||19910409000000|M|||Xyz Hostel^Dtu Campus^Rohini^Delhi^110085^India||||||||||||||||||N
NK1|1|Patient^NextKin|FTH^FATHER||9999123456^^^^91||C^EMERGENCY CONTACT
AL1|1|DA^Drug Allergy|^Penicillin
AL1|2|DA^Drug Allergy|^Sulfa
DG1|1||111735^Diabetes Mellitus Type II, Controlled, with no Complications^L||20150409000000|W
IN1|1|0|UA1|UARE INSURED, INC.|||||||20110101|20260101
```

Figure 9 : ADT A28 HL7 Sample Message

5.4 ORU – R01 HL7 Message

An ORU event R01 message, as shown in Figure 10 is used to transmit lab results to other systems. An ORU message coupled with a medical dictionary can be used to send various medical observations such as weight, height, glucose level, RBC, WBC etc. Using a medical dictionary, we can even send medication information in ORU messages. For example, through OpenMRS dictionary we can send medication information with the help of drug name (also defined in the dictionary), start date and stop date concepts. Therefore, by using the combination of these messages we can transfer information like patient demographics, diagnosis, emergency information, allergies, insurance information, lab test results, medication, and prescriptions data to other systems.

5.5 FHIR Medical Information Sharing Standard

Fast Healthcare Interoperability Resources (FHIR, pronounced "Fire") specify a set of resources that are used to depict granular medical concepts. These resources can be

maintained by some system in isolation, or they can be lumped together into complex documents [38]. FHIR is designed for the web, i.e., the resources are based on simple XML or JSON structures, with an http-based RESTful protocol where each resource has a fixed URL that can be used to get, post, put or delete a resource. It makes use of open internet standards for data representation in these resources.

```
MSH|^~\&|GNU HEALTH|PQR HOSPITAL|JAVA_APP|QUERY_SYSTEM|20140323000000|HUP|ORU^R01|LAB AND MEDICATION|P|2.5|1|||AL||ASCII
PID|||2222^^^GNUHEALTH||Patient^Demo
PV1||O|1^PQR Hospital||||1^PQR Doctor|||||||||||||||||||||||||||||||||||20140323000000|||||||V
OBR|1|||1114^VITAL SIGNS^L||||||||||||1^Doctor^PQR
NTE|1|L|GENERAL OBSERVATIONS (NURSE)
OBX|1|NM|5089^WEIGHT (KG)^L||82|kg|20-300||||F|||20140323000000||2^Nurse^PQR
OBX|2|NM|5090^HEIGHT (CM)^L||174|cm|10-228||||F|||20140323000000||2^Nurse^PQR
OBX|3|NM|5088^TEMPERATURE (C)^L||32.7|DEG C|25-43||||F|||20140323000000||2^Nurse^PQR
OBX|4|NM|5087^PULSE^L||90|rate/min|0-230|H|||F|||20140323000000||2^Nurse^PQR
OBR|2|||1019^COMPLETE BLOOD COUNT^L||||||||||||1^Doctor^PQR
NTE|2|L|ONCOLOGY (TECHNICIAN)
OBX|1|NM|1015^HEMATOCRIT^L||48|VOL%|40-50||||F|||20140323000000||3^Technician^PQR
OBX|2|NM|21^Hemoglobin^L||11.0|g/dl|12.0-16.0|L|||F|||20140323000000||3^Technician^PQR
OBX|3|NM|729^PLATELETS^L||3|10*5/mm3|4-10|L|||F|||20140323000000||3^Technician^PQR
OBX|4|NM|851^MEAN CORPUSCULAR VOLUME^L||87|fl|80-100||||F|||20140323000000||3^Technician^PQR
OBX|5|NM|1017^MEAN CELL HEMOGLOBIN CONCENTRATION^L||32|%|32-37|H|||F|||20140323000000||3^Technician^PQR
OBX|6|NM|1018^MEAN CORPUSCULAR HEMOGLOBIN^L||26|pg|27-33|L|||F|||20140323000000||3^Technician^PQR
OBX|7|NM|678^WHITE BLOOD CELLS^L||4|10*3/mm3|4-10||||F|||20140323000000||3^Technician^PQR
OBX|8|NM|679^RED BLOOD CELLS^L||4.1|10*6/mm3|4.5-6.2|L|||F|||20140323000000||3^Technician^PQR
OBX|9|NM|1016^RED CELL DISTRIBUTION WIDTH^L||11.6|%|11.5-14.5||||F|||20140323000000||3^Technician^PQR
OBR|3|||160741^Medication History^L||||||||||||1^Doctor^PQR
NTE|3|L|MEDICATION (PHARMICIST)
OBX|1|CE|1282^Medication orders^L||72156^BEXAROTENE^L||||||F|||20140323000000||4^Pharmicist^PQR
OBX|2|DT|1190^Historical drug start date^L||20140323000000||||||F|||20140323000000||4^Pharmicist^PQR
OBX|3|DT|1191^Historical drug stop date^L||20140406000000||||||F|||20140323000000||4^Pharmicist^PQR
```

Figure 10: ORU R01 HL7 Sample Message

Some of the resources defined by FHIR are [51]:

- Patient - It includes administrative and demographics information about a patient in question
- RelatedPerson - It includes information about a person that is involved in the care of the patient but is note the target of the healthcare service
- Practitioner - The resource represents information about an individual who provides the healthcare service such as physician, pharmacist, nurse etc.

- Medication - This resource is primarily used for identification and description of certain medicine
- Observation - It include measurements or other assertions made about a patient. They are a central part of healthcare and are at the core of other resources like DiagnosticReport. It can be used to track patient's progress, support diagnosis made by the physician, carry demographics information and determine patterns. For example, observations can be anything from vital signs like blood pressure, temperature etc. to more detailed laboratory data like Complete Blood Count or Thyroid test.
- DiagnosticReport - The resource includes diagnostic test's findings and meanings.
- Encounter - It represents an interaction between a patient and healthcare practitioner.

5.6 Mirth Connect

Mirth Connect is an open source tool that is widely used for transforming non standard data into standard format such as HL7 [52]. It allows routing of standard messages between multiple systems over various transfer protocols like TCP/IP, HTTP and SMTP. The software helps in building HL7 interfaces to non standard compliant applications. It connects two systems using a channel that can fetch data from some source, transform it according to rules set for the channel, filter information and then send it to the destination. All transactions in Mirth Connect can be stored in an internal database so as to group multiple data requests or to archive messages. It allows one to create interfaces using JavaScript facilitating easy transformation and transfer of information.

Mirth Connect supports sending and receiving healthcare messages over a variety of protocols:

- TCP/MLLP
- Database (MySQL, PostgreSQL, Oracle, Microsoft SQL Server, ODBC)
- File (local file system and network shares)
- PDF and RTF documents
- JMS
- FTP/SFTP
- HTTP and SMTP

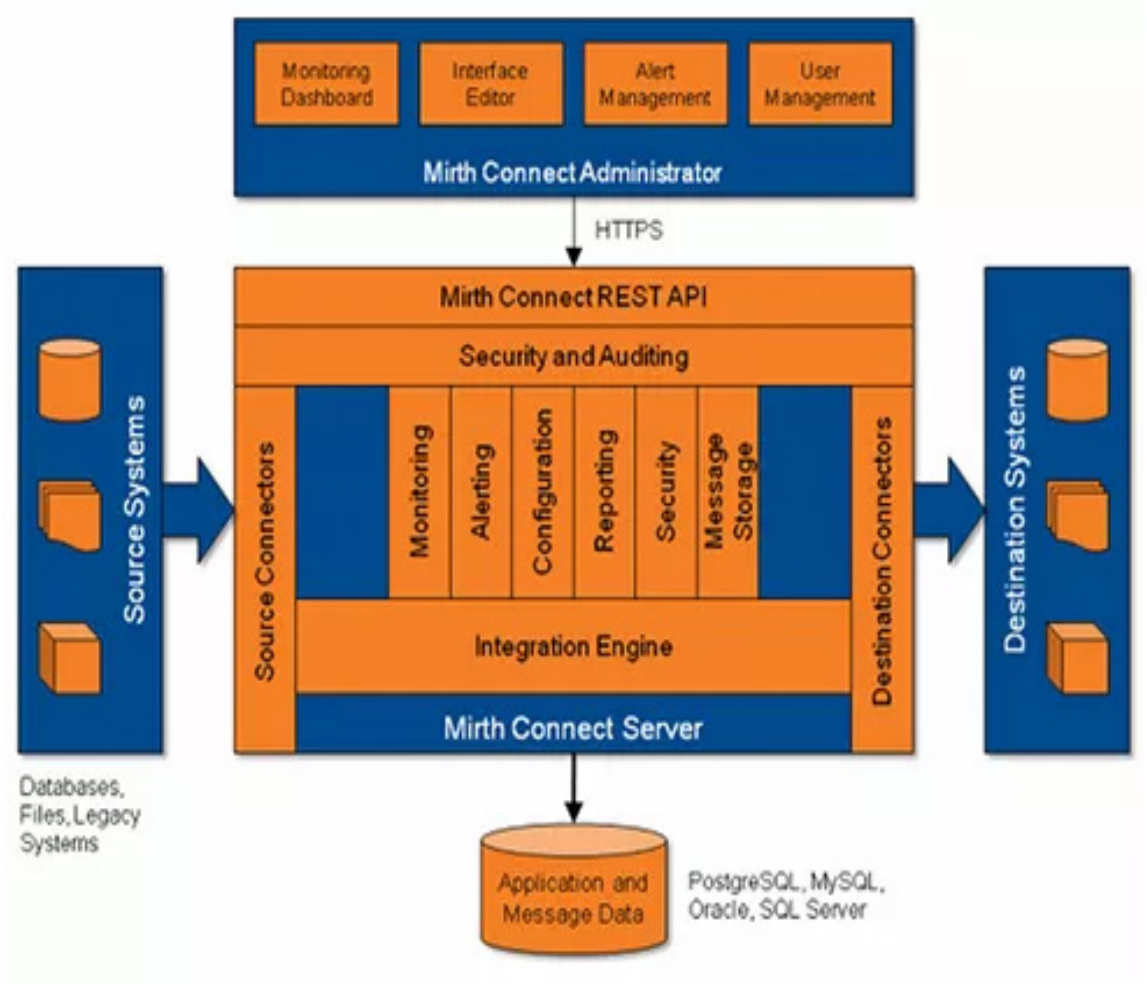

Figure 11 : Mirth Connect Architecture

CHAPTER 6: HADOOP DISTRIBUTED FILE SYSTEM

Apache Hadoop is a Java based open source software framework widely adapted for distributed data storage and processing and was inspired by Google File System and their MapReduce. The platform is composed of following major components [53]:

- Hadoop Distributed File System (HDFS) - a distributed file system the stores data in chunks on inexpensive hardware
- MapReduce - a distributed data processing model that uses Map and Reduce concepts together to analyze large data
- Hadoop YARN - a resource management tool used for assigning resource to users' jobs
- Hadoop Common - a set of useful libraries and utilities used be other components of the platform

HDFS is a highly fault tolerant distributed file system that can be hosted on economical hardware infrastructure [54]. It allows for high throughput access for large data sets by providing streaming access to file system data. HDFS assumes that hardware is bound to fail, thus quickly it quickly discovers faults and provides automated recovery. It is designed to be used as batch processing rather than interactive processing as the emphasis is on high throughput rather than low latency for accessing data. It follows a write once and read anywhere model for files which mean that reads are faster than writes, simplifying data coherency problems. It also works on the fact that moving computation near the data is advantageous than moving data near the computational part which certainly holds true for large data sets. Since, HDFS is built using Java; the file system is platform independent and can run over any platform distribution.

6.1 Components of HDFS

HDFS consist of following components working together:

- NameNode - NameNode is the central part of the distributed system. It manages the name system, i.e., all the directories and files by keeping a directory tree and keeps a track of all the blocks that together manifests a particular file. These blocks are

present on the DataNodes and in no case any file data is stored on the NameNode. HDFS Client always communicates with the NameNode by requesting any operation on a file and NameNode returns a list of DataNodes that are holding the data. NameNode behaves as the master of the system and in earlier releases, it was used to be Single Point of Failure for the file system but efforts are being made to make HDFS more highly available by removing such single point of failure [55].

- DataNodes - They are the systems that are deployed on machines which are supposed to hold the actual data. DataNodes are also known as slaves in the distributed file system. They store file data in blocks and are responsible for serving read and write requests for the HDFS client. A DataNode gives periodic heartbeat signals to the NameNode so as to signify whether it is alive and operational. After HDFS client retrieves the location of a file from the NameNode, it can directly communicate with DataNodes to retrieve file blocks in parallel.
- Secondary NameNode is responsible for performing periodic checkpoints. In the event of NameNode failure, you can restart the NameNode using the checkpoint. The secondary NameNode's job is not only to be a backup for the primary NameNode, but it periodically reads the filesystem changes, log them and applies them into the image file, thus bringing it up to date. This allows the NameNode to start up quicker next time [56].
- HDFS Client - HDFS client interacts with NameNode and DataNode on behalf of user to fulfil user request. User establishes communication with HDFS through File System API and normal I/O operations, processing of user request and providing response over it is carried out by File System API processes.

Figure 12 depicts the overall architecture of HDFS and how HDFS client interacts with NameNode and DataNodes.

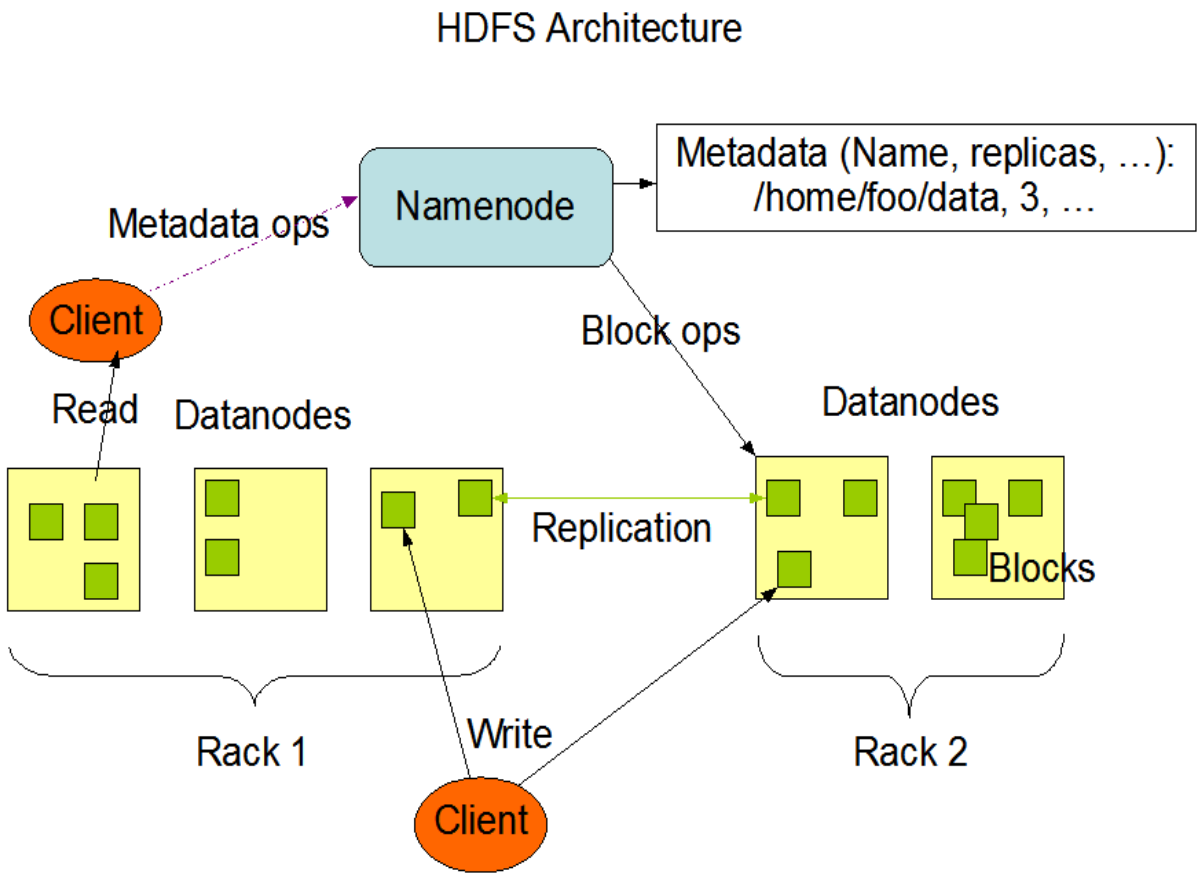

Figure 12 : HDFS Architecture [54]

6.2 HDFS File Operations

The latest version of HDFS supports following file operations:

- Write
- Read
- Append (in later versions)

6.2.1 HDFS Write

The write operation in HDFS is done in seven steps as shown in Figure 13.

1. Create new file in the NameNode's Namespace and calculate block topology
2. Stream data to the first DataNode
3. Stream data to the second DataNode in the pipeline

4. Stream data to the third DataNode
5. Success/Failure acknowledgement
6. Success/Failure acknowledgement
7. Success/Failure acknowledgement

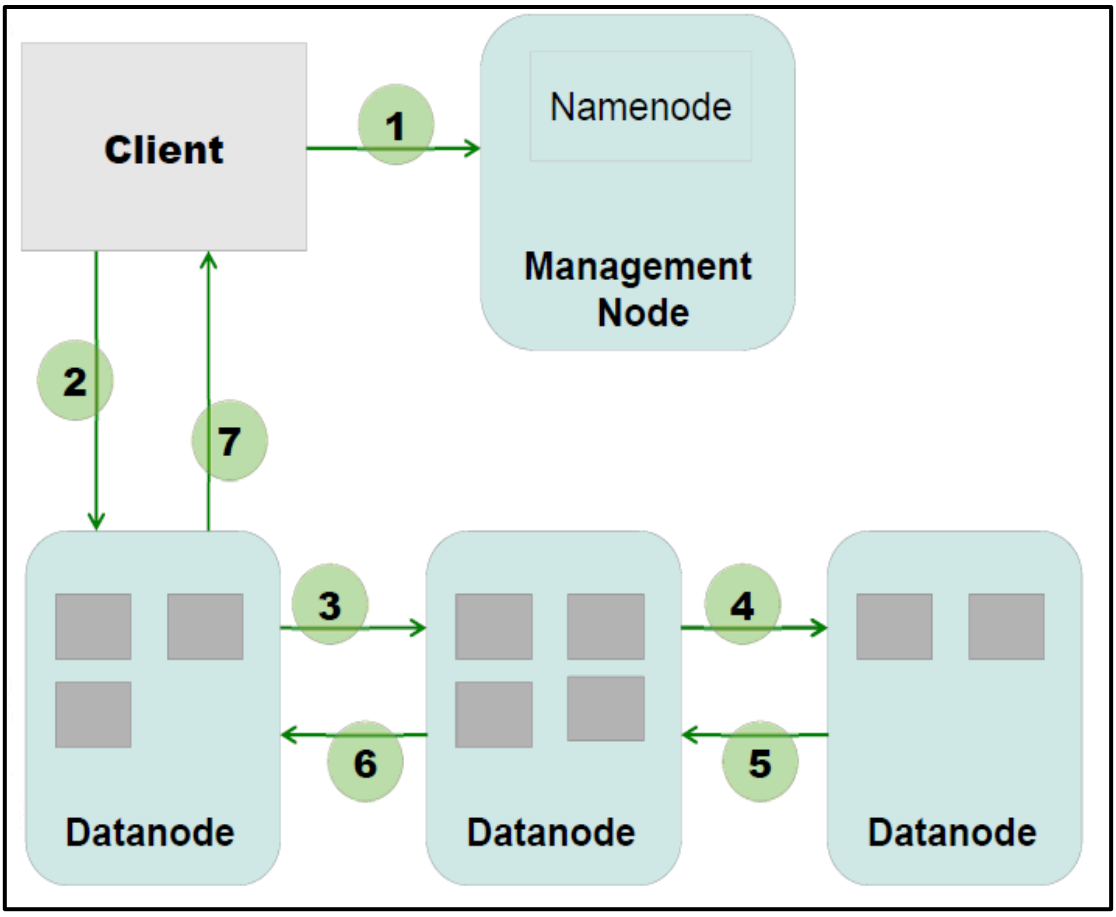

Figure 13 : HDFS Write [57]

6.2.2 HDFS Read

The read operation in HDFS is done in three steps as shown in Figure 14.

1. Client retrieves block location from NameNode
2. Client read blocks to re-assemble the file
3. Client read blocks to re-assemble the file

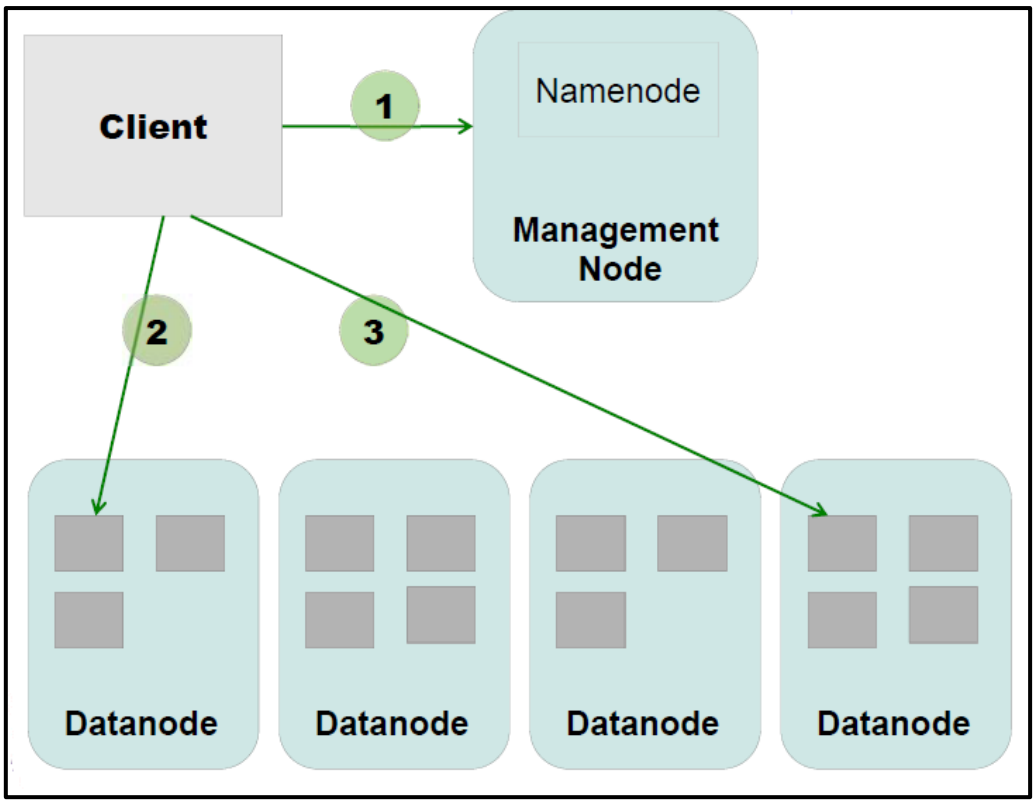

Figure 14 : HDFS Read [57]

6.2.3 HDFS Append

In recent versions of HDFS, append operation has been added to the system. Before the addition of append operation, files on HDFS, once closed, were used to be immutable. For append operation to work, HDFS needs to keep last block of unclosed file visible to the client. This raises two challenges:

- Read consistency. At a given time different replicas of the last block may have different number of bytes. What read consistency should HDFS provide and how to guarantee the consistency even in case of failures.
- Data durability. When any error occurs, the recovery cannot simply throw the last block away. Instead the recovery needs to preserve at least the appended bytes while maintaining the read consistency.

The append operation takes place in the following way:

1. Client sends an append request to NameNode.
2. NameNode checks the file and makes sure that it is closed. Then it checks the file's last block and if it is not full and has no replica, append operation fails. Otherwise, changes are

made to the file. If the last block is full, NameNode allocates a new last block to the file. If the last block is not full, NameNode changes this block as an under construction block, with its finalized replicas as its initial pipeline

3. A pipeline for append operation is setup if last block is not full. Otherwise, one for creating a new file is setup

4. If the last block does not end at a checksum chunk boundary, read the last partial crc chunk. This is for the purpose of calculating checksums.

5. The rest is the same as a regular write.

6.3 MapReduce Paradigm

As defined in [58], "*MapReduce is a framework used for distributed data processing using the well-known MapReduce programming idea. Programs written in this functional style are by design parallelized and executed on a large cluster of commodity hardware. The runtime system takes care of partitioning of the input data, scheduling of the program's execution across a set of machines, handling machine failures, and managing the required inter machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system*."

In the HDFS MapReduce paradigm, each job (or program) has a user-defined map phase which is a parallel, share-nothing processing of input; followed by a user-defined reduce phase where the output of the map phase is aggregated. Typically, HDFS is the storage system for both input and output of the MapReduce jobs.

The main components of MapReduce are as described below [56]:

- JobTracker – It is the master of the MapReduce system that manages the jobs as well as the resources in the cluster (TaskTrackers). The JobTracker schedules each map as close to the actual data being processed or to be used by the application, i.e., on the TaskTracker which is running on the same DataNode as the underlying block on which data resides.
- TaskTrackers - TaskTrackers are the slaves in the MapReduce system and are deployed on each machine involved. They are responsible for running the separate map and reduce tasks as instructed by the JobTracker.

- Job History Server - It is a daemon that serves historical information about completed applications. Typically, Job History Server can be co-deployed with JobTracker

Figure 15 provides details of the core components for the included in MapReduce engine.

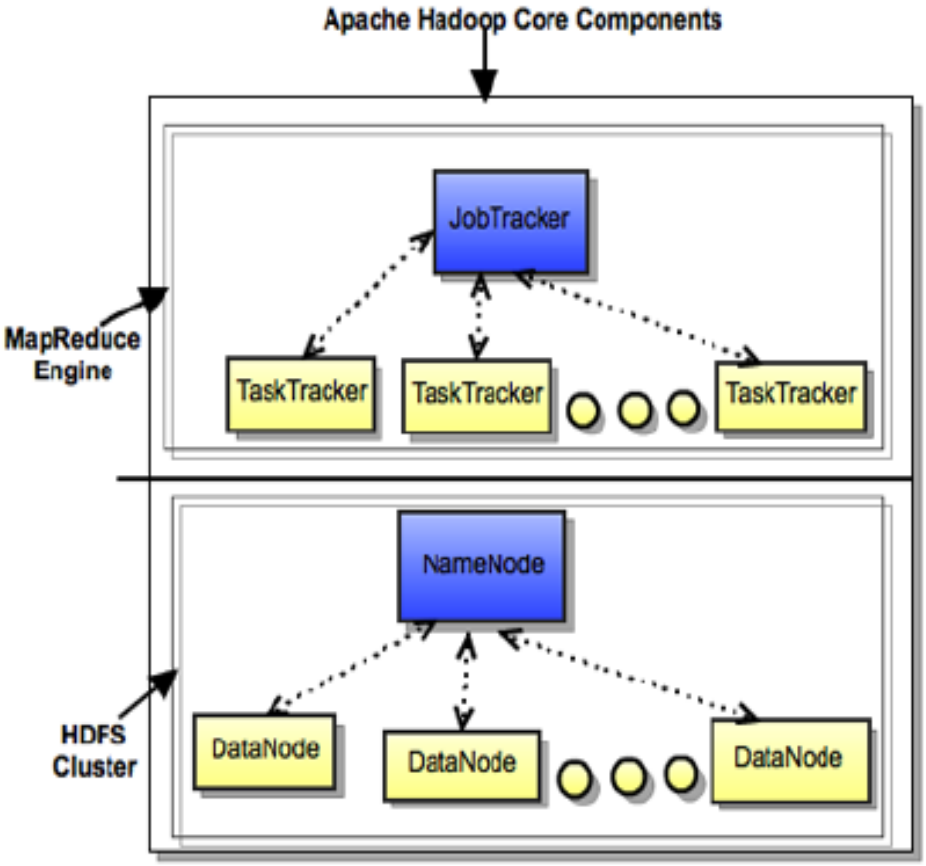

Figure 15 : MapReduce Engine [56]

6.4 HDFS SequenceFiles

HDFS does not provide acceptable performance when working with small files [59] because each file will take a single block of size larger than the file's actual size and for each will have an entry in NameNode that will take actual physical memory that may be greater than the file size. Also the map task in MapReduce processes a block of input at a time, thus there will be more number of map tasks doing little work. Sequence files are nothing but flat container files that consist of binary key/value pairs. It is largely used in MapReduce as input and output formats. Generally, small files are kept together in a single container sequence file with key as file name and value as file contents. Furthermore, Sequence files are splittable, so

MapReduce can break them into chunks and operate on each chunk independently. They also support compression on both record and block levels. Even though SequenceFiles offer many advantages for handling small files it still suffers from some drawbacks. Creating a single SequenceFile from many small files is a hefty task that takes a long time and huge amount of memory. Due to its binary structure, it does not allow appending of records to a single file contained in the big SequenceFile and we need to overwrite the whole SequenceFile to edit single file content.

Figure 16 : HDFS SequenceFile [59]

CHAPTER 7: DESIGN AND IMPLEMENTATION

The work assumes that the primary PHR of the patient resides on a portable device such as a mobile device [1]. In this architecture we provide overview of how backup can be retained on the cloud which can offer to reconstruct the healthcard in case of theft or loss of healthcard and also access some of the medical information through Hadoop Distributed File system for high availability, reliability and response time. The section describes the overall architecture, different components involved along with their tasks and an overview of implementation of our work.

7.1 System architecture

The entire system, as shown in Figure 17, has three main components: a data query and integration system, a middle layer of HL7 interface, and individual hospital information systems storing records in a distributed manner. The system can be used by clinical administrator to provide a backup for the complete medical record of an individual that can then be handed over to the person through a portable device.

Each local HIS stores medical data about patients who have an encounter with a physician affiliated with that hospital. This medical data may encompass information like patient's demographics, current vitals, medication history, prescriptions, diagnosis information, lab test results etc. They keep health records in an electronic format according to their information model and may provide an external interface for retrieval and update of records over the network. Each HIS is responsible for maintenance and availability of health records and it also ensures correctness of medical information stored.

Since these HISs may not share a common data model, health information of a patient gets distributed into heterogeneous data sources. The records pulled from these source needs to be translated into a common format so that they can be assimilated together while maintaining interoperability standards so as to support easy communication and data exchange with different healthcare professionals. HL7 is a feasible medical standard as it provides a common structure to medical information while being flexible enough so that separate HL7 string messages assembled together can represent complete medical history of the patient, while keeping the storage requirement to bare minimum.

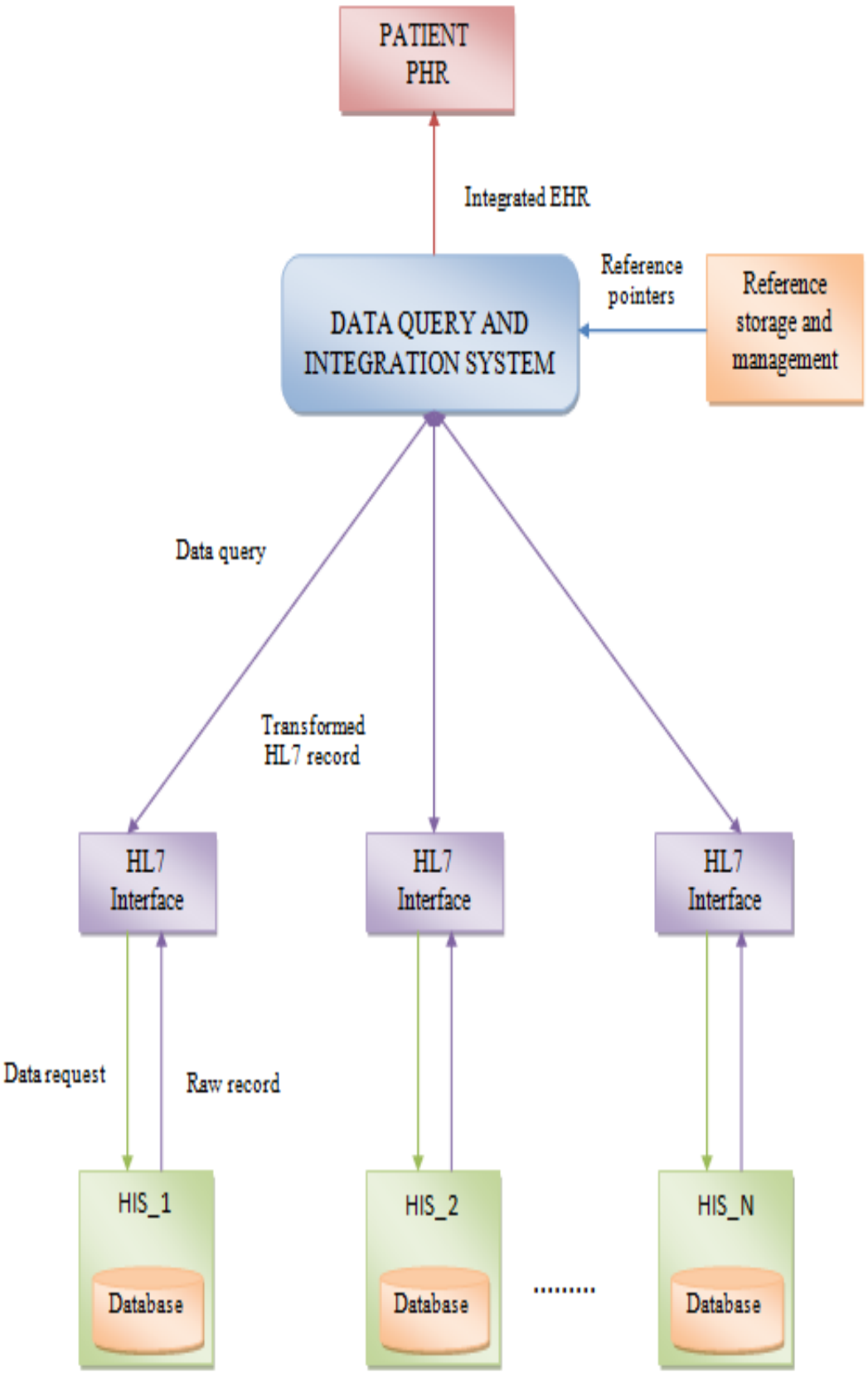

Figure 17 : System architecture for distributed record management system

To retrieve all the medical records of a patient, we maintain a list of reference pointers at a centralized component. These pointers are used by the data query and integration system to request data from each HIS having patient's record. These pointers are maintained in reference file created for each patient registered in the system identified by an external patient identifier. As data received in response from different sources may not be in a standard format, we use HL7 interface layer to forward each query and then get back responses in standardized HL7 layout. All of the HL7 messages received by the system can be encapsulated together into a container format like JSON, representing the integrated EHR, and forwarded to the administrator. The integrated EHR with some pre-parsing of HL7 data can be used to realize the PHR of the patient. The administrator then yields control of the record to the individual who keeps the record on his personal device ensuring confidentiality of records while enabling him to keep track of his medical information.

7.2 Reference Storage and Management

Reference pointers to all the HIS along with files such as lab reports, prescriptions and medications are stored on Hadoop Distributed File System to provide scalable, reliable storage in case of theft or damage to healthcard. A typical reference may include information necessary to track a particular visit stored in the EHR of a healthcare organization like facility name, facility's HIS URL, facility's provider information and patient local identifier. All references for a particular patient are kept as JSON objects encapsulated into a single file identified using a single global patient identifier. Whenever a request comes for regeneration of healthcard for a particular patient, he must provide his global identifier to the system supervisor. He shall query the system to check whether references for that patient are present or not. If a file is present, references to be used for restoration of the PHR are parsed and returned to the user and then queries are made to the subsequent HISs. These reference files are to be made highly available so that medical information can be retrieved as quickly as possible from different sources. For this purpose, we use HDFS that replicates the records so that even system goes down in the cluster, references can still be traced. Using SequenceFiles on HDFS, we are bound to have faster response times than plain File System. Thus, HDFS allows for more reliable and faster access to patient records using the reference model.

7.3 Integrated EHR

The new format proposed will hold all the ORU messages along with a single ADT message encapsulated in a JSON object. The object will have following key-value pairs:

- ADT – HL7 string message containing patient personal information
- ORU – JSON array containing multiple ORU messages each representing a visit

ORU array can be sorted on the basis of timestamp so that most recent visits are above the old ones in the array.


```
{
    "ADT": "HL7ADT patient information message",
    "ORU": [
        {"oru1": "HL7 observation message (visit1)"},
        {"oru2": "HL7 observation message (visit2)"},
        {"oru3": "HL7 observation message (visit3)"},
        .
        .
        .
        {"oruN": "HL7 observation message (visitN)"}
    ]
}
```


Figure 18 : JSON based Integrated EHR

7.4 Implementation

For implementation of the system in our local environment, we have used open source hospital information systems: OpenMRS, OpenEMR and GNU Health for simulating storage of health records at different locations but the system can use any HIS that supports export of patient's medical data.

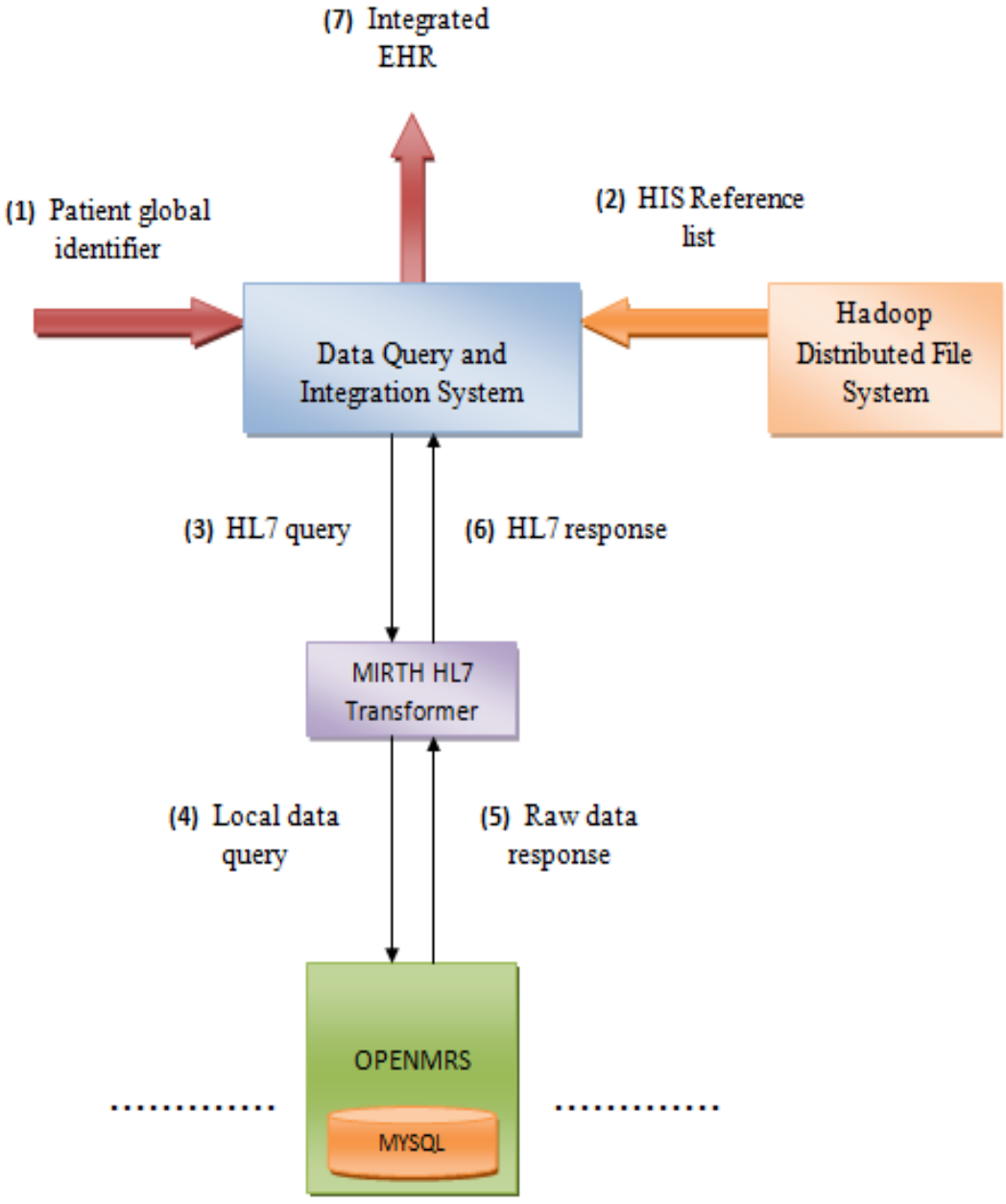

Figure 19 : Overall Workflow with all the components of the system

All of these three systems are built on different technologies and use different data models to store patient records. Both OpenMRS and GNU Health provide REST web services for retrieval of data in non HL7 format but there is no such provision in OpenEMR.

We use Mirth Connect to build external HL7 interfaces to these data sources, enabling us to transform raw data into standard HL7 data. We connect to OpenEMR using a Database Connector so as to retrieve raw data as requested and then transform into HL7 records. Mirth Connect can also send these transformed records over the network to a desired destination. Each of these transformed records may signify an encounter or a visit to a healthcare professional and they are represented as a combination of ADT A28 and ORU R01 messages in HL7.

An overall workflow is given in Figure 18, in which the system first takes patient identifier as an input along with administrator username and password. The data query system then retrieves the encrypted reference file stored on HDFS and decrypted it using ABE. These references are then forwarded to HL7 interfaces built using Mirth Connect. Each interface then retrieves data from the mentioned source and translates it into appropriate format. The transformed message is then routed back to the system and integrated together inside a structured JSON. This JSON file is then handed back as the integrated healthcard of the individual.

Figures 19 – 22 depicts the implemented workflow in which the proposed system has been built in Java which can fetch references from the HDFS, relay those references to Mirth and then integrate the responses received from it together. Mirth Connect is responsible for querying individual HISs and then translating raw responses into HL7 responses and then forward them to the Java based system. The user interface of the system is built using Java Swings and communication with HDFS and Mirth happens over TCP/IP sockets.

In Figure 20, we show the system's authentication screen that requires the registered administrator's username and password along with the global unique identifier of the patient.

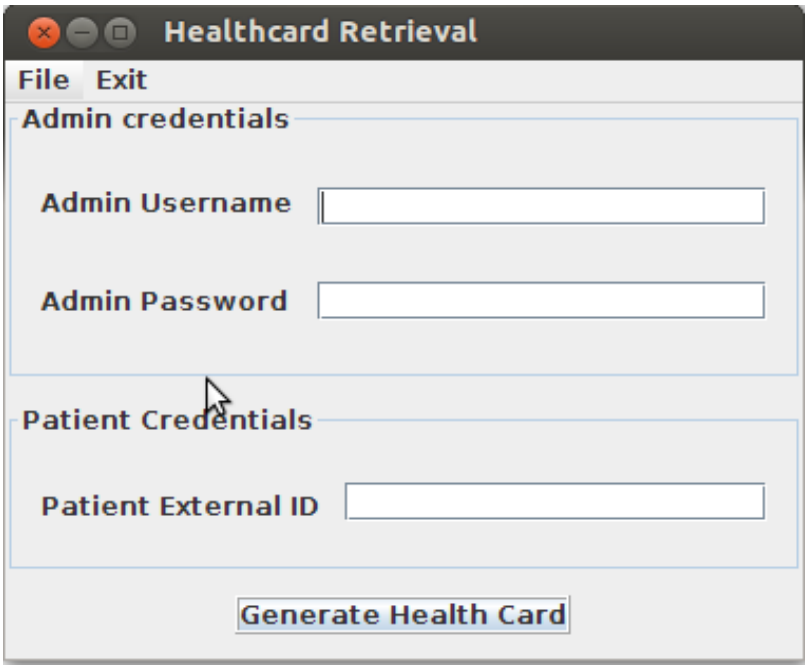

Figure 20 : System Authentication Screen.

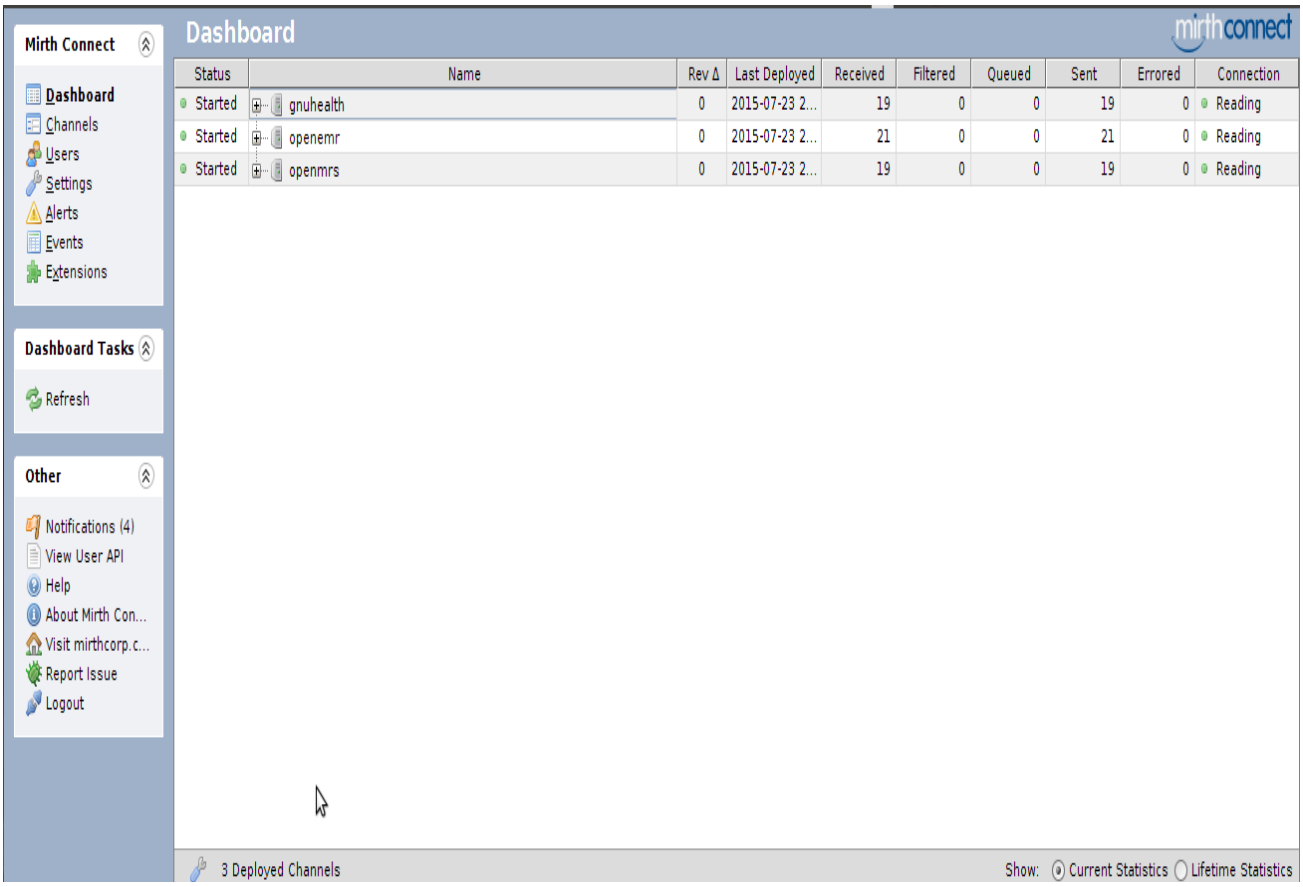

Figure 21 : Mirth Receiving requests and sending translated responses

Once correct information is entered, events are triggered that first retrieve references from HDFS through sockets and then relay them to Mirth, again through sockets and wait for responses from it. Mirth, as shown in Figure 21, queries the referenced HISs in parallel and then transform responses based on the scripts written in JavaScript using E4X standard

Figure 22 describes how the system waits for the data from Mirth and integrates the data together as an encapsulated JSON file including a single ADT message representing personal information of the patient and array of ORU messages representing the visit data.

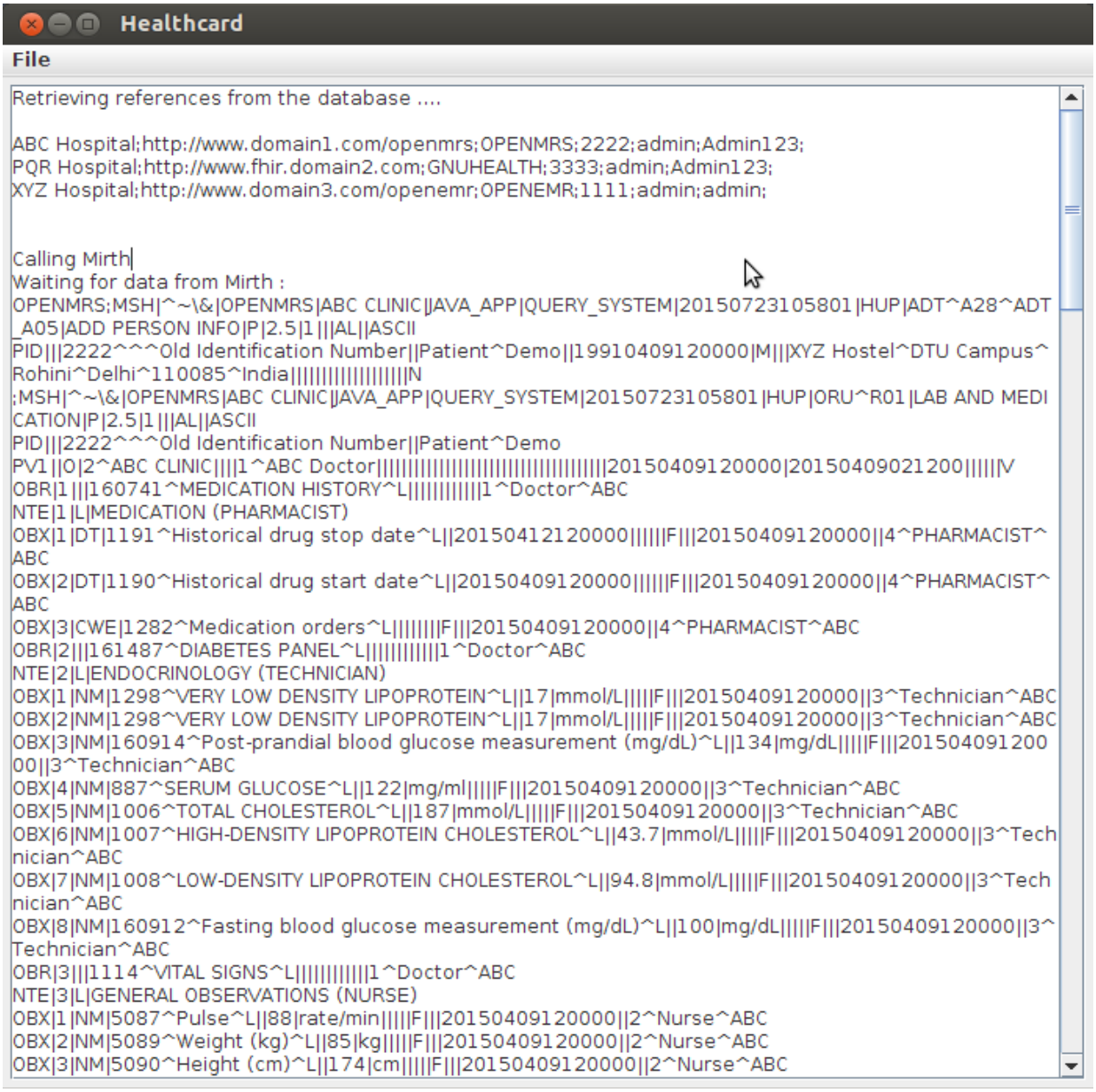

Figure 22 : Responses received by the querying system

In Figure 23, we see how the admin can export the PHR received in parts over the network and then integrated together and forwarded to the patient.

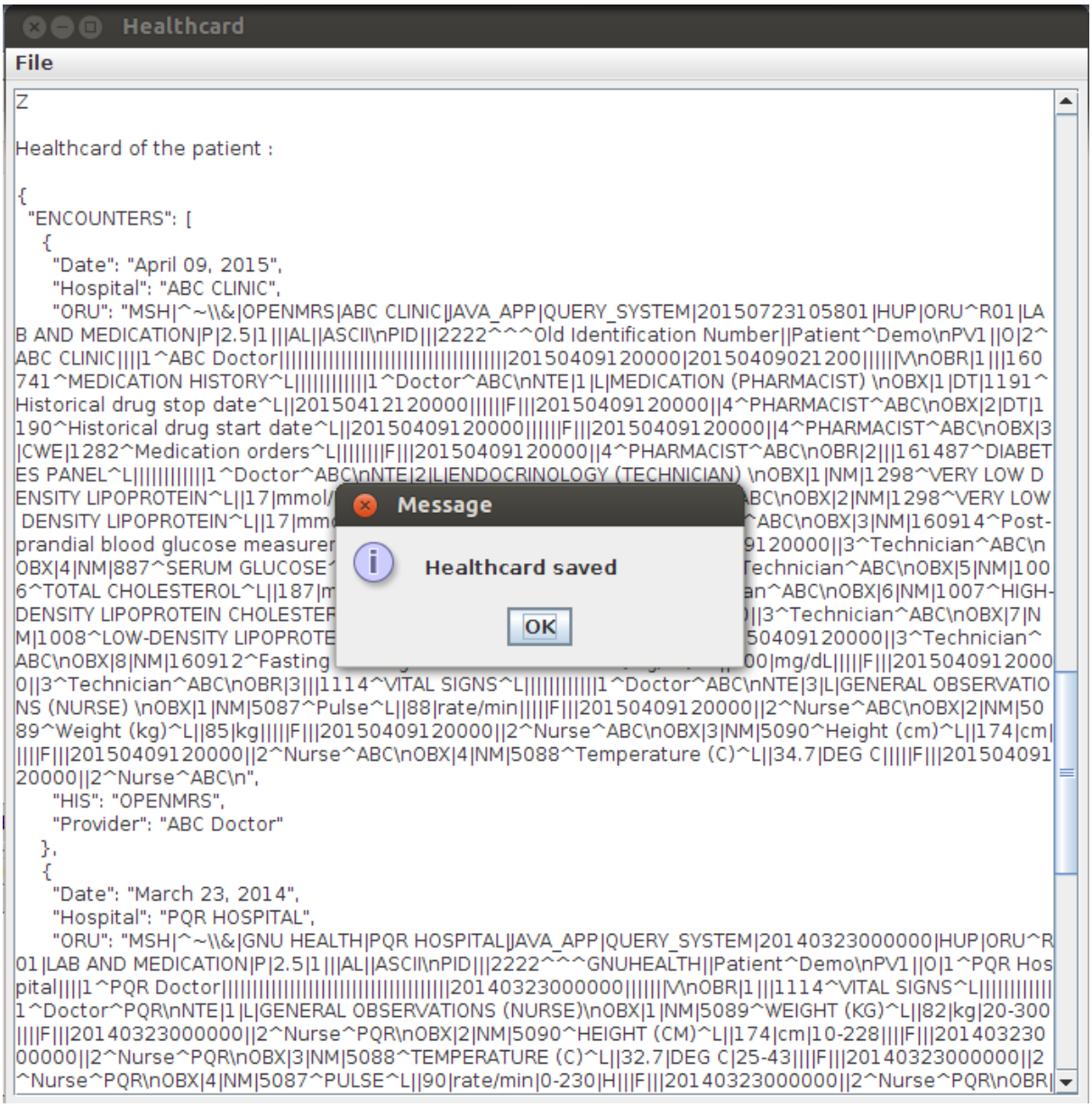

Figure 23 : Healthcard exported as a JSON file

CHAPTER 8: RESULTS

The results of the implementation of the proposed system in a simulated environment indicates that it is successfully able to regenerate the healthcard/ PHR of the patient that is distributed in various hospital systems using only references stored on the distributed file system. HDFS provided high availability through replication and provide decent response times using SequenceFiles.

We tested the availability of HDFS by reading references through Java multithreaded application created with Java version 1.7. Firstly we tried with file of 1MB size for testing the availability and response time of reference files when retrieved in a pseudo concurrent environment. As HDFS does not work well with small files, we used sequence file that shall contain all the reference files for all the patients so as to improve performance [59]. The Hadoop cluster that we tested on consisted of a NameNode with 2 GB of RAM and a DataNode with same amount of memory.

Table 1 shows the time taken to access patient references when files of size 1 MB are stored on local file system and HDFS. Time taken to access references stored in HDFS using SequenceFiles is less compared to those stored in Local file system in regular format.

Table III Local File System Vs HDFS for SequenceFile

No. of files accessed	**Time taken by Local File System (ns)**	**Time taken by Hadoop DFS (ns)**
1	13654402	8244926
10	293183063	64200978
100	7505942552	661442233

Table 2 shows time taken when file size is of 2GB size. In case of large files, time taken to access references stored in HDFS is large as compared to Local file but we are bound to get better results with more volume of data and better hardware infrastructure as reading performance scales better for large set of data [60].

Table IV Local File System Vs HDFS for Large Files

No. of files accessed	**Time taken by Local File System (ns)**	**Time taken by Hadoop DFS (ns)**
1	33877723615	197278325721
10	106077535258	17470047772381
100	643332935076955	1759844631510245

CONCLUSION AND FUTURE WORK

The personal health record of a person allows him to actively monitor his health while giving him direct control over his clinical information. A Patient visiting different healthcare facility will have his data distributed among multiple heterogeneous HISs while he keeps a copy of the record that gets consolidated into his PHR. Although the hybrid PHR model allows confidentiality of data but if, somehow, the healthcard is lost it becomes very complex to restore the PHR. The distributed mechanism discussed provides a secure way to manage and create a backup of the PHR with the help of trusted HISs. The reference model enables the integration of information when required and by keeping these references on the HDFS, it allows for their high availability even on economical hardware.

Through implementation and testing in simulated environment, we can see that the proposed system successfully recreates the PHR of the individual in a reliable manner but realizing it in a real environment do pose certain challenges. Some of the major issues include trustworthiness of different components used and compliance of medical standards by the different healthcare organizations. In a country like India where private sector governs the healthcare industry and sharing of medical data is limited among competing organizations, it becomes essential that the control over the information is in the hands of the patient and he can allow sharing of data with other health professionals while also enabling him to watch over his complete medical history.

We plan to add pushing of references to the central system either by the PHR or by individual HIS. The system also faces certain technical challenges like appending of records in sequence files [59] and improving response times for large media files. We plan to use Apache HBase [61] for improving performance for reading and writing Big data and also to provide better response times for small files in addition to allowing appending of data. It also seems that healthcare community is rapidly shifting to FHIR standard for interoperability as it offers combined advantages of HL7v2 and HL7v3 while focusing on implementation [38]. We also plan to replace HL7v2 based integrated model with FHIR based model in the future.

REFERENCES

[1] Sethia, Divyashikha, et al. "NFC based secure mobile healthcare system."*Communication Systems and Networks (COMSNETS), 2014 Sixth International Conference on*. IEEE, 2014.

[2] Sprague, Lisa. "Personal health records: the people's choice." *NHPF Issue Brief* 820 (2006): 1-13.

[3] Tang, Paul C., et al. "Personal health records: definitions, benefits, and strategies for overcoming barriers to adoption." *Journal of the American Medical Informatics Association* 13.2 (2006): 121-126.

[4] Brian Raymond. "Realizing the Transformative Potential of Personal Health Records." Kaiser Permanente Institute for Health Policy, In Focus, v1, 2007.

[5] Detmer, Don, et al. "Integrated personal health records: transformative tools for consumer-centric care." *BMC medical informatics and decision making* 8.1 (2008): 45.

[6] Wuerdeman, Lisa, et al. "How accurate is information that patients contribute to their electronic health record?." *AMIA Annual Symposium Proceedings*. Vol. 2005. American Medical Informatics Association, 2005.

[7] Kaelber, David C., et al. "A research agenda for personal health records (PHRs)." *Journal of the American Medical Informatics Association* 15.6 (2008): 729-736.

[8] Spine Services — Health and Social Care Information Centre, *http://systems.hscic.gov.uk/spine*

[9] Campion-Awwad, Oliver, Alexander Hayton, Leila Smith, and Mark Vuaran." The National Programme for IT in the NHS.", 2014.

[10] van't Noordende, Guido. "Security in the Dutch electronic patient record system." *Proceedings of the second annual workshop on Security and privacy in medical and home-care systems*. ACM, 2010.

[11] Force, G. E. T. "Electronic health records: A global perspective Second Edition." Technical report, *HIMSS Enterprise Systems Steering Committee*, 2010.

[12] Daglish, David, and Norm Archer. "Electronic personal health record systems: a brief review of privacy, security, and architectural issues." *Privacy, Security, Trust and the Management of e-Business, 2009. CONGRESS'09. World Congress on*. IEEE, 2009..

[13] Borthakur, Dhruba. "HDFS architecture guide." *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf* (2008).

[14]Krohn, Rick. "The consumer-centric personal health record—it's time." *Journal of Healthcare Information Management* 21.1 (2007): 21-23.

[15]Halamka, John D., Kenneth D. Mandl, and Paul C. Tang. "Early experiences with personal health records." *Journal of the American Medical Informatics Association* 15.1 (2008): 1-7.

[16]Official Google Blog - An update on Google Health and Google PowerMeter, *http://googleblog.blogspot.in/2011/06/update-on-google-health-and-google.html*

[17]Sunyaev, Ali, Dmitry Chornyi, Christian Mauro, and Helmut Krcmar. "Evaluation framework for personal health records: Microsoft HealthVault vs. Google Health." *In System Sciences (HICSS), 2010 43rd Hawaii International Conference* on, pp. 1-10. IEEE, 2010.

[18]Agarwal, Ankur, Borko Furht, and Vivek Tyagi. "Integration of Various Health Record Systems." *Handbook of Medical and Healthcare Technologies*. Springer New York, 2013. 503-528.

[19]Koppar, Anant R., and V. Sridhar. "Tele-Health Medical Diagnostics System with Integrated Electronic Health Records." *Indian Journal of Public Health Research & Development* 3.3 (2012): 9-13.

[20]Yu, Wang, et al. "Design and Development of a Clinical Data Exchange System Based on Ensemble Integration Platform." *Frontier and Future Development of Information Technology in Medicine and Education*. Springer Netherlands, 2014. 385-392.

[21]Huba, Nicholas, and Yan Zhang. "Designing patient-centered personal health records (PHRs): health care professionals' perspective on patient-generated data." *Journal of medical systems* 36.6 (2012): 3893-3905.

[22]Baldock, Daniel James, et al. "Electronic health record sharing using hybrid architecture." U.S. Patent No. 8,650,045. 11 Feb. 2014.

[23]van't Noordende, Guido. "Controlled dissemination of electronic medical records." *Proceedings of the 2nd USENIX conference on Health security and privacy. USENIX Association*. 2011.

[24]EMR vs EHR – What is the difference?, *http://www.healthit.gov/buzz-blog/electronic-health-and-medical-records/emr-vs-ehr-difference/*

[25]EMR vs EHR vs PHR, *http://ed-informatics.org/healthcare-it-in-a-nutshell-2/emr-vs-ehr-vs-phr/*

[26]Haux, Reinhold. "Health information systems–past, present, future."*International journal of medical informatics* 75.3 (2006): 268-281.

[27]Goldschmidt, Peter G. "HIT and MIS: implications of health information technology and medical information systems." *Communications of the ACM*48.10 (2005): 68-74.

[28]OpenMRS, *http://openmrs.org*

[29]Concept Dictionary Basics, *https://wiki.openmrs.org/display/docs/Concept+Dictionary+Basics*

[30]Seebregts, Christopher J., et al. "The OpenMRS implementers network."*International journal of medical informatics* 78.11 (2009): 711-720.

[31]RESTful Web Service, *http://www.ibm.com/developerworks/webservices/library/ws-restful/*

[32]Restful Crud Operation on a WCF Service, *http://www.codeproject.com/Articles/115054/Restful-Crud-Operation-on-a-WCF-Service*

[33]Getting and Using the MVP-CIEL Concept Dictionary, *https://wiki.openmrs.org/display/docs/Getting+and+Using+the+MVP-CIEL+Concept+Dictionary*

[34]OpenEMR Project, *http://www.open-emr.org/*

[35]OpenEMR Features, *http://www.open -emr.org/wiki/index.php/OpenEMR_Features*

[36]GNU Health – Freedom and Equity in Healthcare, *http://health.gnu.org*/

[37]GNU Health Introduction, *https://en.wikibooks.org/wiki/GNU_Health/Introduction*

[38]Introducing HL7 FHIR, *http://www.hl7.org/fhir/summary.html*

[39]FHIRREST server, *https://en.wikibooks.org/wiki/GNU_Health/FHIR_REST_server*

[40]Using the FHIR REST server, *https://en.wikibooks.org/wiki/GNU_Health/Using_the_FHIR_REST_server*

[41]Eichelberg, Marco, et al. "A survey and analysis of electronic healthcare record standards." *Acm Computing Surveys (Csur)* 37.4 (2005): 277-315.

[42]What is OpenEHR, *http://www.openehr.org/what_is_openehr*

[43]Understanding the Continuity of Care Record, *http://www.corepointhealth.com/sites/default/files/whitepapers/continuity-of-care-record-ccr.pdf*

[44]HL7 Version 2 Product Suite, *http://www.hl7.org/implement/standards/product_brief.cfm?product_id=185*

[45]CDA Release 2, *http://www.hl7.org/implement/standards/product_brief.cfm?product_id=7*

[46]The HL7 Clinical Document Architecture,
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC130066

[47]An overview of CCD templates, *http://www.hl7standards.com/blog/2008/07/23/an-overview-of-ccd-templates/*

[48]Comparing HL7 messages to HL7 documents,
http://www.hl7standards.com/blog/2008/01/25/comparing-hl7-messages-to-hl7-documents/

[49]HL7 Sample Message**,** *http://www.altova.com/HL7_technology_primer.html*

[50]HL7 Resources, *http://www.corepointhealth.com/resource-center/hl7-resources*

[51]FHIR Resource Index, *http://www.hl7.org/fhir/resourcelist.html*

[52]Mirth Connect, *http://www.mirth.com/Products-and-Services/Mirth-Connect*

[53]Apache Hadoop*, https://en.wikipedia.org/wiki/Apache_Hadoop*

[54]Borthakur, Dhruba. "HDFS architecture guide." *HADOOP APACHE PROJECT http://hadoop. apache. org/common/docs/current/hdfs design. pdf* (2008).

[55]HDFS NameNode, *http://wiki.apache.org/hadoop/NameNode*

[56]Apache Hadoop core components,
http://docs.hortonworks.com/HDPDocuments/HDP1/HDP-1.2.4/bk_getting-started-guide/content/ch_hdp1_getting_started_chp2_1.html

[57]Hadoop Tutorial:
Developing Big-Data Applications with Apache Hadoop,
http://www.coreservlets.com/hadoop-tutorial/

[58]Dean, Jeffrey, and Sanjay Ghemawat. "MapReduce: simplified data processing on large clusters." *Communications of the ACM* 51.1 (2008): 107-113.

[59]The Small files problem, *http://blog.cloudera.com/blog/2009/02/the-small-files-problem/*

[60]*Dinh, Tien Duc. "Hadoop Performance Evaluation." (2009).*

[61]Apache HBase, *http://hbase.apache.org/*

APPENDIX

A - Writing a SequenceFile made from reference files on HDFS through Java

```
package writeFile;

import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.Text;

public class WriteData {

	int num_records;

	public WriteData(int num_records) {
		this.num_records = num_records;
	}

	public void SeqFileWrite()throws IOException{
		Configuration conf = new Configuration();
		FileSystem fs = FileSystem.get(conf);
		FSDataInputStream inputStream;
		Text key = new Text();
		Text value = new Text();
		org.apache.hadoop.io.SequenceFile.Writer.Option
filePath = SequenceFile.Writer
				.file(new Path("/refs.seq"));
		org.apache.hadoop.io.SequenceFile.Writer.Option
keyClass = SequenceFile.Writer
				.keyClass(Text.class);
		org.apache.hadoop.io.SequenceFile.Writer.Option
valueClass = SequenceFile.Writer
				.valueClass(Text.class);
```

```
          SequenceFile.Writer writer =
SequenceFile.createWriter(
                    conf, filePath, keyClass, valueClass);

          for (int i = 1; i <= 1000; ++i) {
               String str = "";
               Path inputFile = new
Path("/Patient_References/ref" + i + ".js");
               System.out.println("Processing file : " +
inputFile.getName() + " and the size is : " +
inputFile.getName().length());
               inputStream = fs.open(inputFile);
               key.set(inputFile.getName());
               while (inputStream.available() > 0) {
                    str = str + inputStream.readLine();
               }
               value.set(str);
               writer.append(key, value);

          }
          fs.close();
          IOUtils.closeStream(writer);
     }

     public static void main(String[]args) {
          try {
               new
WriteData(Integer.parseInt(args[0])).SeqFileWrite();
          } catch (IOException e) {
               // TODO Auto-generated catch block
               e.printStackTrace();
          }
     }
}
```

B - Reading N number of reference files from a SequenceFile made on HDFS through Java

```
package readFile;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.SequenceFile;
import org.apache.hadoop.io.SequenceFile.Reader.Option;
import org.apache.hadoop.io.Writable;
import org.apache.hadoop.util.ReflectionUtils;

public class ReadFile {

	int num_records;

	public ReadFile(int num_records) {
		this.num_records = num_records;
	}
	private void readSequenceFile(String
sequenceFilePath)throws IOException {
		// TODO Auto-generated method stub

		/*
		 * SequenceFile.Reader sequenceFileReader = new
SequenceFile.Reader(fs,
		 * new Path(sequenceFilePath), conf);
		 */
		Configuration conf = new Configuration();
		FileSystem fs = FileSystem.get(conf);
		Option filePath = SequenceFile.Reader.file(new
Path(sequenceFilePath));
		SequenceFile.Reader sequenceFileReader = new
SequenceFile.Reader(conf,
				filePath);

		Writable key =
(Writable)ReflectionUtils.newInstance(
			sequenceFileReader.getKeyClass(), conf);
		Writable value =
(Writable)ReflectionUtils.newInstance(
			sequenceFileReader.getValueClass(), conf);
```

```
            try {

                  long startTime = System.nanoTime();
                  for (int i = 0; i < num_records; ++i) {
                        sequenceFileReader.next(key, value);
                        System.out
                        .printf("[%s] %s %s \n",
                              sequenceFileReader.getPosition(),
key,
                              value.getClass());
                  }
                  long endTime = System.nanoTime();
                  System.out.println(endTime - startTime);
            }
            finally {
                  IOUtils.closeStream(sequenceFileReader);
            }

      }

      public static void main(String[]args) {
            ReadFile rf = new
ReadFile(Integer.parseInt(args[0]));

            try {
                  rf.readSequenceFile("/refs.seq");
            } catch (IOException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
      }

}
```

C - Deploying GNU Health on local environment

Requirements:

1. Operating System: GNU Health is Operating System independent, although we are using Ubuntu Distribution 14.04

2. Database: PostgreSQL

3. Python: >= 2.7 < 3.0

4. Tryton: = 3.4

5. BASH Shell

6. PIP version for Python 2

Step I. Install dependencies

```
#apt-get install build-essential python-dev python-pip \
libxml2-dev libxslt1-dev libldap2-dev libsasl2-dev python-ldap \
python-imaging python2.7-cracklib postgresql postgresql-server-dev-all
```

Step II. Creating the Operating System User

Run the following command as root:

```
useradd -m gnuhealth
```

If you already created the "gnuhealth" operating system user, you can skip this section, otherwise, create it now.

Step III. Verify PostgreSQL authentication method

PostgreSQL uses different authentication methods (MD5, ident, trust). Depending upon the Operating System, the PostgreSQL server authentication method will vary. The standard GNU Health installation uses the trust authentication method, so you need to check the PostgreSQL authentication file configuration. Locate the pg_hba.conf file and verify that the

trust method is set. The location of this configuration file can be obtained with the following command, to be executed as root:

su - postgres -c "psql -t -P format=unaligned -c 'show hba_file'"

An example configuration file entry specifying use of the trust method is given in the following line:

local all all trust

After any changes to the file, the PostgreSQL server needs to be restarted.

Step IV. Downloading and Installing GNU Health

Do the following steps with your newly created gnuhealth user and do not use root:

1. Become user gnuhealth:

```
su - gnuhealth
```

2. Download GNU Health from GNU.org

```
wget http://ftp.gnu.org/gnu/health/gnuhealth-latest.tar.gz
```

3. Uncompress the file:

```
tar xzf gnuhealth-latest.tar.gz cd gnuhealth-*
```

4. Run the gnuhealth_install.sh script

./gnuhealth_install.sh

5. Finally, enable the BASH environment for the gnuhealth user.

source "$HOME"/.gnuhealthrc

Step V. Activate Network Devices for the JSON-RPC Protocol

The Tryton GNU Health server listens to localhost at port 8000, not allowing direct connections from other workstations.

```
editconf
```

Edit the parameter listen in the [jsonrpc] section, to activate the network device so workstations in your net can connect. Add the following block:

```
[jsonrpc]

listen = *:8000
```

This will allow connecting to the server in the different devices of your system.

Step VI. Setting up a Local Directory for Attachments

By default, Tryton uses a system-wide directory to store the attachments. In GNU Health it is advisable to keep the attachments in the gnuhealth user space. Edit the server configuration file trytond.conf and enter the attach directory under the [database] section, for instance:

```
editconf

[database]

path = /home/gnuhealth/attach
```

Step VII. Booting up the Tryton Server

Change to your newly installed system (use the alias cdexe)

```
cdexe
```

Boot the server using the command

```
./trytond
```

Step VIII. Installation of the Tryton Client

Download and extract the Tryton client from the tryton site:

http://downloads.tryton.org/3.4/tryton-3.4.2.tar.gz

Install python dependencies:

```
pip2 install --user python-dateutil
```

Untar the client tarball:

```
tar -xzvf tryton-3.4.2.tar.gz
```

Excute the client

```
cd tryton-3.4.2/bin
```