PEACH Streams: Defining an OpenEHR Service Implementation with Stream processing for Computational Healthcare Cases in Radiology

ANDRE ALLORERUNG, University College London
ABHIMANYU CHAKRABARTY, University College London
CONNOR CLOONEY, University College London
NARMIN HUSEYNLI, University College London
EFTHYMIA KAZAKOU, University College London
DEAN MOHAMEDALLY, Univesity College London, Supervisor
NAVIN RAMACHANDRAN, University College London Hospital, Stakeholder

Having knowledge of the technological problems the National Health Service (NHS) is experiencing with systems integration and data sharing services development, we propose PEACH Streams, an OpenEHR platform which aims to provide stream processing services for various Healthcare-related cases in Radiology. It intends to scale with IaaS expansion as new systems are prototyped, integrated and trialled with the benefits of enabling external systems to utilise a neutral architectural layer. By providing health data adaptors and connectors for demographic querying as well as statistical analysis, it facilitates data consistency, availability and resilience. Based on our findings, we evaluated our system as a proof of concept in terms of scalability. The main advantage of this project is the transformation and aggregation of such data in real time using the IaaS which was not available before. Moreover, the architecture of the infrastructure allows smooth component integration and evolution.

Additional Key Words and Phrases: OpenEHR, PEACH Streams, Stream Processing, Radiology

1. INTRODUCTION

Improving technology and rising standards of healthcare services raise the public expectation for an efficient, integrated and user-friendly care service. The NHS currently lacks a suitable infrastructure for day to day activities throughout the different NHS regions. The current implementation does not provide a proper channel for secure interoperability between said regions. Consequently, while there are plenty of applications for healthcare services, they are not used extensively due to lack of a secure and consistent platform, to which domain specific applications can interface with to provide the necessary data and experience to the end users.

The infrastructure needed will serve not only to formalise a healthcare exchange by connecting a substantial number of clinicians across medical institutions, but also provide care workers and patients with availability and subscriber access to healthcare systems. The vision for PEACH Streams is for an infrastructure that supports research, education and practice progression of NHS systems. As the demand for a wide variety of classifications of domain specific implementations grow, the applications developed can hook on to and utilise the PEACH backbone.

Through PEACH Streams, the ability to manage, translate and transmit a pipeline of data across multiple systems components enables a user to classify and contain a pathway of data based on a healthcare case. This by design is a development pattern to support future streams as the infrastructure and availability of applications grows with PEACH. Early results of PEACH Streams have been added in a scientific paper aimed to be submitted to applicable conferences [Allorerung et al. 2016].

The main contributions of this paper will be:

(1) Implementing the foundation of an open access healthcare infrastructure based on a modular, secure and consistent architectural layer.
(2) Providing a data processing pipeline which regulates, transforms, and transfers healthcare-related data across multiple domains.

1.1. Motivating Example

Let us assume that User A wants to utilise a healthcare infrastructure for research, education and practice purposes so that he can capture, transform and transfer data across multiple systems. The NHS structure however does not support such a reliable and coherent solution: no secure nor resilient infrastructure is provided.

In this example, based on the modularity and flexibility provided by our proposed infrastructure, PEACH Streams will provide User A with a secure and easy access feature to all the existing applications as well as with adaptors and connectors to integrate new applications into it. From the composition of the system, User A will be able to create his own encrypted data flows based on different healthcare-related cases along different regions of the NHS. In addition to that, our system will provide him with multiple alternatives to manipulate the incoming data for further processes such as analytics and machine learning services. Finally, we aspire PEACH to continuously evolve in which the user can easily expand the existing architecture to fit his needs.

The rest of the paper is organised as follows: Section 2 presents the background work; Section 3 shows the requirements and analysis of the proposed system; Section 4 describes the design and architecture of our system approach; Section 5 provides the proposed evaluation of this approach shows our results and impact; Section 6 presents the best practices and Section 7 offers concluding comments and suggestions for future work.

2. BACKGROUND

2.1. Literature Review

The literature in software engineering, applied computer science and medical engineering has helped define the starting point for our work on the PEACH infrastructure. There is an extensive body of research in the wide area of computer-based healthcare management, including its interaction with analytics, big data and discussions of system architecture. This literature is too broad to review here, but several key points are important to highlight:

2.1.1. Big Data in healthcare. In the last decade, Big Data has grown as a prominent player in almost every sector. Recent studies also show this phenomenon is currently occurring in healthcare and its related areas. [Shakshuki et al. 2015] for example, has conducted research on how data streaming is utilised via daily health or medical devices and sensors, pointing out existing challenges in the lifecycle: scalability, heterogeneity, timeliness, and security.

A number of recent studies have discussed and experimented on the challenges: [Rallapalli et al. 2016] and [Ojha and Mathur 2016] talk about the scalability and [Celesti et al. 2016] tries to tackle the problem of heterogeneity. However, a major issue of timeliness is one of the biggest concerns of today's Big Data. PEACH infrastructure is heavily made based on a study of a similar case by [Ta et al. 2016], with the suggestion of an infrastructure that is built on the foundation of Lambda architecture [Vanhove et al. 2016]. It provides stream processing of data but still preserves the batch processing path, shown by the compound Apache Kafka, Apache Storm, Apache Hadoop, HBase and Apache Cassandra. By creating the stream path, data can be processed in real time. This is one of the best approaches to the timeliness issue, however it can be simplified, e.g. by moving the design to Kappa architecture or omitting complex components such as Apache Storm from the stack and use other simpler or built-in components. PEACH infrastructure also looks deeper into this matter of deciding the most efficient architecture and what technologies realise it, as will be discussed further in the next sections.

2.1.2. Extract-Transform-Load (ETL) pipeline. The term ETL is one of the cornerstones in the world of Big Data. ETL pipeline activities may vary from case to case. [Vassiliadis et al.] gives unary activities and n-ary activities as the example. A simple input-transform-output with only a single agent at the starting point and another at the end point is considered as an unary activity. However, PEACH stream pipeline should be viewed as a complex n-ary activities, where it ingests raw data from a set of applications, performs complex transformation by using more than one schema or a schemata group, and pushes the output to a set of consumers. Although the activities seems to be a spider's web, one can also simplify the activities by viewing them as series of binary activities. PEACH infrastructure is built with the focus on a stream processing form of an ETL pipeline. However, since the current phase of development is not fully concentrated on the perfection of real-time data processing and latency, the development considers stateful and near-streaming technologies to be utilised, most notably Apache Samza and Apache Spark.

2.1.3. Online Analytical Processing (OLAP) services. OLAP is a powerful technology for the applications to purposefully utilise data in order to evaluate the current situation and bring ease into decision making for further iterations [Sethi]. As opposed to a relational database, OLAP tools expand the two-dimensional analysis to multidimensional space which allows a user to estimate business activity by logically aggregating the variables to allow for analysis from a different point of view. The following basic OLAP analytical operations can be enumerated to consolidation, drill-down, and slicing/dicing [OBrien and Marakas 2010]. The modification of OLAP technology is Real Time Online Analytical Processing (RTOLAP). This approach enables analytics on live streams and performs calculations instantly as the data is available in the RTOLAP system. As a real-time OLAP tool, Druid has been included in the PEACH infrastructure to provide flexible exploratory data analyses with low latency data injection [Dehne and Zaboli 2012].

2.2. Related Work

In the relation to our proposed infrastructure, a number of researches, experiments, and product releases are related closely to PEACH Streams, or share similar instruments, techniques, and methods with our infrastructure. Those projects also have contributed to the ideas of our proposed PEACH Streams.

As the first example, a "Patient management" component of the *RadCentre operational system* has been utilised over the last decades by the Radiology Department of UCLH which is employed to facilitate authorised users-clinicians with executing several operations over the medical records. Alongside aforementioned features, the users are able to monitor patient attendances, obtain full audit trails as well as being provided with customisable letters, reports and labels in a printed version via electronic health records. However, this application couldn't handle real-time data processing as a part of an e-health infrastructure that PEACH aims to provide.

Neftlix's Keystone data-pipeline example which is considered as one of the best practices that have been evolved throughout the years leveraging various architectural styles [Blog 2016a], [Blog 2016b]. Netflix engineers and data scientists are handling billions of events per day and are driving actionable insights from the data analysis instigated. The data experiences several transformations until building the latest version of a Keystone Data pipeline which is a real-time stream processing system offering petabyte scalability for business and product analysis.

NorthWest EHealth (NWEH) has also developed a technology platform with a Linked Database System (LDS) approach, instead of standard Randomised Controlled Trials, to obtain trial data instantly from its source. The design principle of LDS and other

supporting services enable the capture of eHealth data from various medical organisations in near real time and utilize these findings in the healthcare research and development [Elkhenini et al. 2015].

VigiLanz is such of the real-time Enterprise Intelligence Resources EIR platforms to boost the health care delivery process by performing clinical intelligence, business intelligence and producing predictions based on real-time medical data. The platform can accept an input in a form of real-time values or collection of the health records gathered throughout years from a variety of resources and provide clinicians with patient diagnostics immediately [Intelligence 2016].

DACAR is another eHealth platform that enables to implementation of cloud-based architecture in a distributed and secure way by introducing roles based and inter-domain policies [Fan et al. 2013]. Serving as a Platform as a service (PaaS), DACAR addresses several issues widely seen in a healthcare domain such as authentication, authorisation, data persistence, data integrity, data confidentiality and audit trail.

3. REQUIREMENTS AND ANALYSIS OF PROPOSED SYSTEM

At the highest level PEACH infrastructure seeks to achieve the following goals:

3.1. High Level Goals

(1) Improve healthcare services provided by NHS.
(2) Reduce public expenditure in the healthcare sector.
(3) Integrate Microsoft technologies with PEACH's open source infrastructure and applications.
(4) Utilise OpenEHR services in order to support OpenEHR compliancy for future development and integration with other applications.

3.2. Requirements

For the purpose of achieving the high level goals, a number of requirements are set. The requirements listed detail the vital points that the infrastructure must have.

(1) PEACH infrastructure must be able to interface with the designated set of applications from various domains. This means it has the proper components that are used to communicate with external systems, no matter how diverse these systems are.
(2) PEACH infrastructure must have a controller or director of data flow, instructing how raw data is received, processed, and given to every requesting component and end point.
(3) The infrastructure must also have the means to transform any kind of data that flows throughout the pipeline, to any desired formats for further processes, user queries and analyses. Hence, PEACH should have a dedicated component that can perform the desired transformation, which for current implementation is the combination of information coming from a number of data streams.
(4) The infrastructure must be able to allow data to be persisted as logs. Logs should be preserved for audits and/or future usage.
(5) As a performance requirement, the transformed data should be ready for consumption, queries and analyses in near real-time. Thus, a dedicated application should also be placed at the end point of the infrastructure with the task of providing a medium for user queries.
(6) Security must be incorporated within PEACH infrastructure. This should be done by ensuring communication channels and the flowing data are secure. An example of it is to utilise end-to-end encryption.

4. DESIGN AND ARCHITECTURE

4.1. System Overview

The infrastructure's main architecture overview is represented as an **adjusted Kappa architecture** pattern [O'Reilly 2016], viewed from the perspective of traditional input-storage-output system. The stack is composed of the following layers:

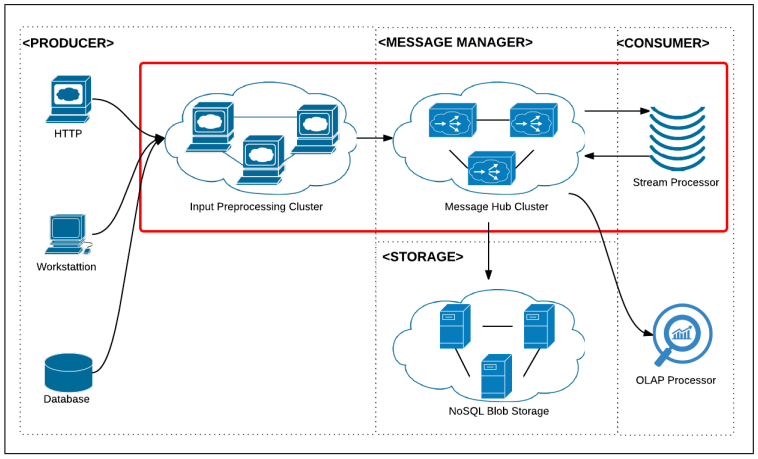

Fig. 1. PEACH Streams Architectural Design.

Producer. The layer consists of a range of data input components for the whole flow of PEACH. Considering that specific forms of data are needed, a pre-processing component is placed, consuming inputs of various file types from the providers and performing transformations before passing it to the message manager layer.

Message Manager. Following the pre-process of the input data, the products will then be given to this publish-subscribe messaging system: queued as messages in topics waiting to be consumed by the components in other layers. The way which data should go into which topic is decided previously by the pre-processing cluster in the producer layer, and each component in the consumer layer has the freedom to choose from which topics the data will be ingested.

Consumer. Every application domain which falls within this layer listens to specific topics in the message hub cluster and retrieves messages from it. A stream processor can also be considered as a consumer, yet it produces new transformed data and sends it back to the message hub to be consumed by other elements in the consumer layer that are listening to the specific topic.

Storage. A cluster of blob storage is used to hold the files and data objects from the message hub as logs, which act as instruments for analysis, auditing, or other processes in the future.

The PEACH Streams' process is maintained by a group of components which forms a data flow pipeline structure, marked in the red box as shown in figure 1. Following the base design in Kappa architecture, the data flows real-time from the entry gate of the infrastructure, throughout the pipeline, and consumed by designated application

domains. Despite that, modifications are made to the channel: the infrastructure provides pre-processing cluster at the starting point, and reconnects the end point of the stream processor back to the message hub.

4.2. PEACH Infrastructure

As part of the specification of our infrastructure, a search has been conducted to explore existing technologies and understand their potential relevance to our project. The technologies incorporated in our infrastructure that is shown in figure 2 are reviewed below.

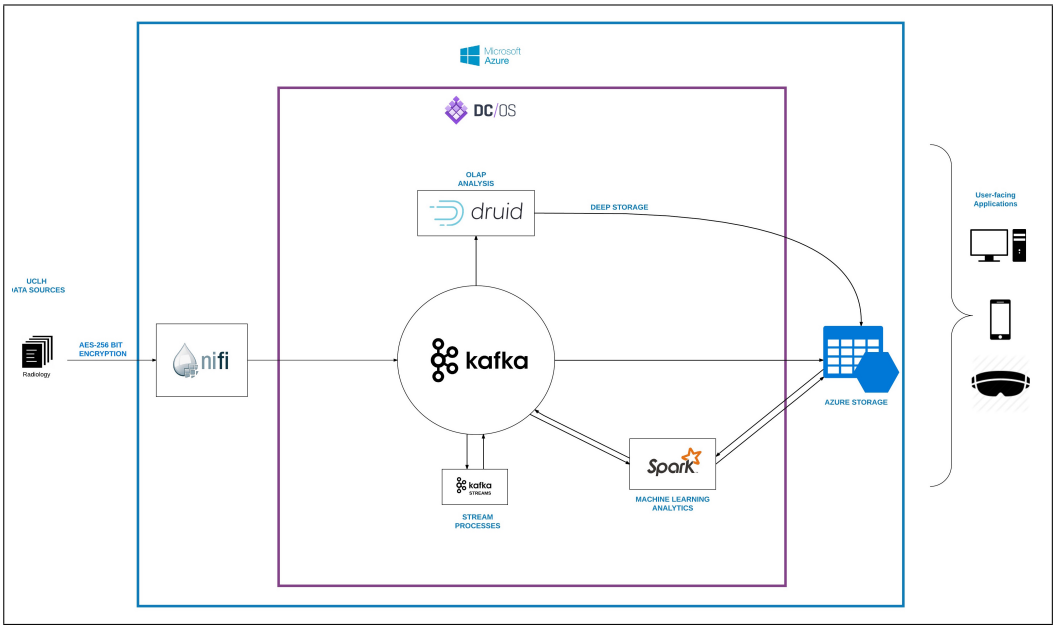

Fig. 2. PEACH Infrastructure.

- *DC/OS*: We used DC/OS template on Azure as the cluster manager operating the PEACH infrastructure. We chose DC/OS because it supports service discovery load balancing and package management and because of its multiple agents it handles points of failure. From a deployment perspective, it can be easily used with the Azure stack and provides some built-in components that are accessible from its graphical user interface [DC/OS 2016][Magazine 2016].

- *Apache Spark*: Apache Spark comprises an important part of our infrastructure since it can perform complex analytics over diverse and very large datasets.For this reason we built connectors from Apache Kafka to Spark and also from Spark to Azure storage such that in the future iterations, all of Spark's features can be used easily for machine learning tasks [InfoQ 2016].

- *Apache Kafka*: We consider Apache Kafka as the central message broker in PEACH system as it extracts data from various inputs and pushes the aggregated data into other subsystems. It is reliable and it has high throughput and good replication functionalities [Kreps et al. 2011].

- *Kafka Streams*: The main purpose of using Kafka Streams [Warski 2016] [Kreps et al. 2011] in PEACH is to perform real time data pre-processing and then push back newly formed topics so that they can be consumed by Druid.

- *Druid*: In PEACH, Druid [Yang et al. 2003] is used as a query and OLAP service on event driven data. It combines a column-oriented storage layout, a distributed, shared-nothing architecture, and an advanced indexing structure. It also allows the arbitrary exploration of billion-row tables with sub-second latencies using slicing and dicing techniques.

- *Apache NiFi*: Our main purpose of using Apache NiFi is for pre-processing the raw radiology data to conform with Druid's guidelines and observe its metadata. It is also used to initiate multiple data pipelines and transfer sensitive data encrypted across diverse platforms [Fox et al. 2015].

- *Microsoft Azure Storage*: We proposed the use of Azure blob storage to act as a deep storage option for non-compacted and unstructured data. [ArrkGroup 2015], [Bhat 2016]. A benefit of using it is having the ability to use Hot and Cold Storage Tiers that allow the user to cost-effectively define the stored data in terms of how often it will be accessed.

4.3. OpenEHR compliancy

The MDT sub-domain, which is described in section A.2.1 of the appendix, within the PEACH project has been developed with the intention of being OpenEHR compliant. The OpenEHR, or open electronic health record, is a standard for the practice of creating digital health data in a consistent uniform format to support interoperability between healthcare systems. A key OpenEHR component used within the project is EtherCIS (Ethereal Clinical Information System) [EtherCIS 2016]. EtherCIS is an open source implementation of an OpenEHR compliant component which is currently in beta.The OpenEHR area in the medical industry is not a well developed area as, although the concept has been present for a few years, there is a lack of implementations that have been adopted on a large scale.

However, there is a large potential gain in the intended case where healthcare establishments are utilising OpenEHR compliant systems. Whereby, the various siloed medical systems and their interoperability issues are removed, improving the ability for healthcare establishments to efficiently operate between medical departments and establishments. Upon request of our client we moved forward with the implementation of the relatively new OpenEHR service that is EtherCIS, where it's only notable usage had previously been with biomedical data [Delussu et al.]. EtherCIS provides a backend service to applications, allowing them to use RESTful APIs to exchange OpenEHR data in the forms of JSON and XML to clients machine and persist the data to relational and PostgreSQL database implementations. These JSON and XML data formats make up the OpenEHR archetypes that are designed to be compatible and usable across any OpenEHR system that is in use by any healthcare establishment.

5. EVALUATION

5.1. Experimental Setup

During the experiment, we initially relied on a reserved Virtual Machine (VM) within UCL Computer Science Department to run all systems and perform testing on the infrastructure. Additionally, our storage system is deployed remotely as an Azure Storage instance. Table I shows our environment specification. In order to setup an appropriate experimental environmemt for our evaluation we ensured the following:

(1) Fully anonymised raw data was extracted from the UCLH radiology database in a CSV format and was sent to the UCL server via a secure channel (ssh) in an AES 256bit encryption. These files were composed of six reference type records

and three user input records that were simulated as real-time data coming into the system every five minutes.

Table I. PEACH Environment Specification

Component	Description
VM Processor	Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz Dual Core
VM Memory	2 GB (2040 MB) RAM
VM Physical Memory	40 GB
VM Operating System	CentOS Linux release 7.2.1511
Storage System	Microsoft Azure Blob Storage

(2) The simulation started from the pre-processing layer, and it went through *three stages* or *groups of processors* before reaching the message hub cluster.

— The **first group** was tasked with breaking the .CSV files into smaller portions since they consisted of one million records and could potentially burden the overall performance of the system. These products were then stored back into the VM's physical space.
— The **second group** handled the majority of data transformations. This included a transformation into JSON format as well as modifications to the content of the files. The modifications were necessary to replace all the invalid fields so that the data could be stored in Druid. A couple of changes were also made to make some attributes more comprehensible. The processor ended by breaking the JSON file into 10,000 JSON objects.
— The **third group** transmitted the JSON objects produced to a specific Kafka topic every second. As a result, we had nine topics in our Kafka cluster, corresponding to the nine CSV files we initially received. The topics were replicated in two Kafka brokers within our VM. Since only three datasets could be simulated as real-time streamed data, we focused on them for the real-time simulation and used the six other topics as static source of information.

(3) The data sent to these three topics are consumed by *Kafka Streams* where more preprocessing is being performed. Subsequently, Kafka Streams performs a big left join operation for all the reference and real time data per examination scan. After the join is completed, Kafka Streams produces a new topic with the joined data that is published to *Druid*.
(4) At this step Tranquility server acted as a middleware and log messages were injected to Druid's storage through a Kafka Tranquility module of Tranquility. Via Druid's web interface we could examine the status of the injection task where "SUCCESS" indicated when a task had been successfully accomplished.
(5) The data topics could also be consumed by *Spark* via its consumer class where it receives the JSON records at an interval of every 1000 ms as a microbatch of RDD streams.

Based on the aforementioned experimental setup, we evaluated our result by comparing the input coming from NiFi and the content in Druid as well as the blob file located in the Azure cloud platform.

5.2. Testing methods

5.2.1. Unit Testing. PEACH's components consist of large interconnected technologies, where some components have minimal customisation and are running as-deployed to provide a service. Hence, there was only a small amount of custom classes/methods to write unit tests for. A particular component that was proved useful to utilise unit tests

was Kafka, where functions for managing data were tested to ensure that they return the expected types of data. Additionally, unit testing with JUnit has been performed against several inputs to test the connection to the Azure Storage account and CRUD operations over the blob objects.

5.2.2. Black box testing. Black box testing was used across the PEACH infrastructure where the test outputs could be validated for individual components as well as test outputs for a sequence of components that processed a piece of data [Ostrand and Balcer 1988]. This was useful when testing the interaction between NiFi and Kafka, by validating the test output against our expected result to assess the transformation process.

5.3. Results

Performance of the system was measured by its processing time. In table II we provide a sample result of this measurement in the pre-processing layer for which we took 10,000 records of the DATAEXAM file which is one of the examined datasets. For simplicity purposes, complexity is measured by counting the number of processors incorporated within the same group. We show at Stage 2, a CSV file pre-processing to singular JSON objects. There we illustrate that a complex transformation is completed in less than 4 milliseconds per record, or just over 37 seconds in total. In Stage 3, the time required to finish a 10,000 record transfer to the message hub cluster is 2 hours and 47 minutes, or less than 1.001 second per record. The 1 millisecond lag is due to our VM memory space but is still tolerable. In another experiment at Stage 3, 10,000 records in a batch were put to stream queue before sending them to Kafka and only approximately 15 seconds were needed for their transformation. Finally, we experimented that the data flow from Stage 2 to Stage 3 without storing data to physical disk but directly pushing it to Kafka was 5 milliseconds per record, or just over 50 seconds total.

Table II. Sample result of measurement in pre-processing layer

Stage	Input		Output		Time	Complexity
	# files	Size (MB)	# files	Size (MB)		
1*	1 (CSV)	494.31	299 (CSV)	494.31	00:00:21.227	13
2	1 (CSV)	1.65	10,000 (JSON)	7.17	00:00:37.419	9
3**	10,000 (JSON)	7.17	10,000 (JSON <key, value>)	7.17	02:46:50.722** 00:00:15.224***	5
2-3	1 (CSV)	1.65	10,000 (JSON <key, value>)	7.17	00:00:50.467	12

* The whole DATAEXAM dataset is broken into smaller CSV files, initially with 2,987,567 records. Following stages use one file of the products which contains 10,000 records.

** The records are simulated to be processed one record per second.

*** The records are simulated to be processed 10,000 records at one time in a queue.

5.4. Impact

Because of PEACH's modular infrastructure, diverse data stream inputs from external dedicated systems or medical IoT devices can be supported for real time data processing. This enables the clients of the system to expand the data handled and also allows for further medical stakeholders to adapt PEACH to have their own data sources sent through the infrastructure. Domain specific applications can be developed to utilise the data services of PEACH, providing access to datasets and the stream data processing functionalities of the infrastructure. This provides future technological expansion

opportunities for the stakeholders, as well as improving the client's digital healthcare resources at present.

6. BEST PRACTICES

During these three months we were exposed to different technologies and techniques. The following are knowledge we acquired and issues which limit the deliverables.

— As a containerised operating system we selected DC/OS for its better documentation and online support, however due to our limited Azure resources, we did not fully explore this path. Thus we hope future works will focus on the allocation of the resource for building DC/OS.
— Apache NiFi is a powerful tool to design, build, and monitor the flow of the data in stream queues. We also encourage the use of NiFi to be widely extended to other domains rather than just in data pre-processing, so that each application may have smaller-sized NiFi framework as its container.
— The use of Kafka in our architecture allows very high throughput and partition aware processing. It is required for use cases such as efficient stream processing where data needs to be reprocessed.
— Java based Kafka Consumer API was used to consume messages published to Kafka brokers and load them continuously to the deep storage before the post processing step. Since the consumer API is not thread-safe [Kafka 2016], we got exposed to apply multi-threaded programming techniques to synchronize access appropriately.
— Azure Blob Storage gave us the chance to work with the Microsoft Azure Storage SDK for Java[ArrkGroup 2015]. However, downsides of this practice can be shown as a lack of services and operations supported in comparison to .NET libraries and available documentation because of its new release.
— Building connectors between Kafka and Spark was challenging because of the dependencies needed for the Spark, Kafka and Scala version. Scala latest version 2.11.8, did not support all the functionalities needed and therefore we had to choose version 2.10.5 along with the compatible Kafka and Spark versions in order to avoid any errors.
— Druid required a timestamp column, dimension and metrics that would assist to rollup raw data once messages loaded to the Druid cluster. Another hurdle was that Druid cluster does not support join operations and thus an ETL stack was needed to perform the join [Yang et al. 2003].

7. CONCLUSIONS AND FUTURE WORK

As a future work we propose further developing the PEACH analytics platform that will include the development of the Spark and Druid integrations. In addition new data sources should be added and new data pipelines should be developed by utilising Apache Kafka Streams processors and Apache NiFi processors. Furthermore, future teams should operate Apache Spark machine learning and R analytics environments as well as deploying the ELK stack and Zeppelin on Spark. In addition to this Druid could be used to potentially interact with User-Facing applications by using analytics tools such as Metabase and Caravel, where the event data stored in Druid segments will be presented using slicing and dicing techniques. Finally, there is a need for developing EtherCIS with use of new database technologies such as VoltDB.

ACKNOWLEDGMENTS

The authors would like to thank its UCLH and Microsoft Stakeholders for their guidance and technical support. They would also like to thank UCL professors and tutors who got involved in this project and helped in its completion.

REFERENCES

Andre Allorerung, Abhimanyu Chakrabarty, Connor Clooney, Narmin Huseynli, and Efthymia Kazakou. 2016. PEACH Streams: Defining an OpenEHR Service Implementation withStream processing for Computational Healthcare Cases in Radiology, Vol. 2.

ArrkGroup. 2015. Microsoft Azure Storage (Blob/Table/Storage) White Paper. http://www.arrkgroup.com/wp-content/uploads/2015/07/Arrk-Group-Microsoft-Azure-Storage-White-Paper-v1.pdf

Sriprasad Bhat. 2016. Azure Blob Storage: Hot and Cool Storage Tiers. (2016). Retrieved June 18, 2016 from https://azure.microsoft.com/en-us/documentation/articles/storage-blob-storage-tiers/

The Netflix Tech Blog. 2016a. Evolution of the Netflix Data Pipeline. (February 2016). Retrieved August 21, 2016 from http://techblog.netflix.com/2016/02/evolution-of-netflix-data-pipeline.html

The Netflix Tech Blog. 2016b. Netflix details Evolution of Keystone data pipeline. (February 2016). Retrieved August 21, 2016 from https://www.infoq.com/news/2016/03/netflix-keystone-data-pipeline

Antonio Celesti, Agata Romano Maria Fazio, and Massimo Villari. 2016. A hospital cloud-based archival information system for the efficient management of HL7 big data. 39th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO) 39th. Croatian Society MIPRO, Opatija, 406–411. DOI:http://dx.doi.org/10.1109/MIPRO.2016.7522177

Mike Cohn. 2005. *Agile Estimating and Planning: Chapter 6 Techniques for Estimating*. Prentice Hall.

Graham Collins. Agile Project Management. In *Appendix 1*. Faculty of Engineering Sciences, University College London.

DC/OS. 2016. DC/OS reference implementation: The Azure Container Service. (2016). Retrieved June 11, 2016 from https://dcos.io/docs/1.7/overview/design/azure-container-service/

Frank Dehne and Hamidreza Zaboli. 2012. Parallel Real-Time OLAP on Multi-Core Processors. 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE. DOI:http://dx.doi.org/10.1109/CCGrid.2012.19

Giovanni Delussu, Francesca Frexia Luca Lianas, and Gianluigi Zanetti. A Scalable Data Access Layer to Manage Structured Heterogeneous Biomedical Data. Pula, CA, Italy. DOI:http://dx.doi.org/:http://dx.doi.org/10.1101/067371.

Hanaa Elkhenini, Kourtney Davis, Norman Stein, Mark Delderfield John New, Martin Gibson, Ashley Woodcock Jorgen Vestbo, and Nawar Diar Bakerly. 2015. Using an electronic medical record (EMR) to conduct clinical trials: Salford Lung Study feasibility. DOI:http://dx.doi.org/10.1186/s12911-015-0132-z

EtherCIS. 2016. EtherCIS an open source openEHR server. (2016). Retrieved August 18, 2016 from http://ethercis.github.io/

L. Fan, W. Buchanan, and C. Thummler. 2013. DACAR Platform for eHealth Services Cloud. IEEE, Cloud Computing (CLOUD), 2011 IEEE International Conference on, New York, NY, 219–226. DOI:http://dx.doi.org/10.1109/CLOUD.2011.31

G. C. Fox, S. Kamburugamuve J. Qiu, S. Jha, and A. Luckow. 2015. HPC-ABDS High Performance Computing Enhanced Apache Big Data Stack. Shenzhen, 1057–1066. DOI:http://dx.doi.org/10.1109/CCGrid.2015.122

InfoQ. 2016. Big Data processing with Apache Spark Part 1:Introduction. (2016). Retrieved June 10, 2016 from https://www.infoq.com/articles/apache-spark-introduction/

Vigilanz Intelligence. 2016. Real time Healthcare Intelligence improves Quality and Patient Safety. (2016). Retrieved August 24, 2016 from http://www.gartner.com/imagesrv/media-products/pdf/Vigilanz/vigilanz-1-2VC2NFK.pdf

Apache Kafka. 2016. Kafka Consumer. (2016). Retrieved August 29, 2016 from https://kafka.apache.org/090/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html

Jay Kreps, Neha Narkhede, and Jun Rao. 2011. Kafka: a Distributed Messaging System for Log Processing. NetDB'11, Athens, Greece.

Redmond Magazine. 2016. Azure Container Service goes live with Mesosphere DC/OS integration. (2016). Retrieved June 11, 2016 from https://redmondmag.com/articles/2016/04/19/azure-container-service-goes-live.aspx

James OBrien and George Marakas. 2010. *Management Information Systems* (10th ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.

Mimoh Ojha and Kirti Mathur. 2016. Proposed application of big data analytics in healthcare at Maharaja Yeshwantrao Hospital. 3rd MEC International Conference on Big Data and Smart City (ICBDSC). In *Muscat*. IEEE, 1–7. DOI:http://dx.doi.org/10.1109/ICBDSC.2016.7460340

O'Reilly. 2016. Questioning the Lambda Architecture. (2016). Retrieved August 31, 2016 from https://www.oreilly.com/ideas/questioning-the-lambda-architecture

T. J. Ostrand and M. J. Balcer. 1988. The category-partition method for specifying and generating functional tests. In *Communications of the ACM*, Vol. 31. New York, NY, USA, 676–686. DOI:http://dx.doi.org/10.1145/62959.62964

Sreekanth Rallapalli, R.R. Gondkar, and Uma Pavan Kumar Ketavarapu. 2016. Impact of Processing and Analyzing Healthcare Big Data on Cloud Computing Environment by Implementing Hadoop Cluster. In *Procedia Computer Science*, Vol. 85. Elsevier B.V, 16–22.

O. Salo and P. Abrahamsson. 2007. Agile methods in European embedded software development organisations: a survey on the actual use and usefulness of Extreme Programming and Scrum. In *Appendix 1*. The Institution of Engineering and Technology 2008, Oulu, Finland. DOI:http://dx.doi.org/10.1049/iet-sen:20070038

Manya Sethi. Data Warehousing and Olap Technology. International Journal of Engineering Research and Applications (IJERA). International Journal of Engineering Research and Applications (IJERA), 955–960. http://www.ijera.com/papers/Vol2_issue2/FD22955960.pdf

Elhadi Shakshuki, Rudyar Cortes, Xavier Bonnaire, Olivier Marin, and Pierre Sens. 2015. Stream Processing of Healthcare Sensor Data: Studying User Traces to Identify Challenges from a Big Data Perspective. Conference on Ambient Systems, Networks and Technologies (ANT-2015), the 5th International Conference on Sustainable Energy Information Technology (SEIT-2015)Stream Processing of Healthcare Sensor Data: Studying User Traces to Identify Challenges from a Big Data Perspective. In *Procedia Computer Science*, Vol. 52. Elsevier B.V, 1004–1009. http://www.sciencedirect.com/science/article/pii/S1877050915008935

Van-Dai Ta, Chuan-Ming Liu, and Goodwill Wandile Nkabinde. 2016. Big data stream computing in healthcare real-time analytics. International Conference on Cloud Computing and Big Data Analysis (ICCCBDA). IEEE, Chengdu, 37–42. DOI:http://dx.doi.org/10.1109/ICCCBDA.2016.7529531

Thomas Vanhove, Gregory Van Seghbroeck, Tim Wauters, and Filip De Turck Bruno Volckaert. 2016. Managing the Synchronization in the Lambda Architecture for Optimized Big Data Analysis. In *IEICE Transactions on Communications 99*, Vol. 2. 297–306.

Panos Vassiliadis, Alkis Simitsis, and Eftychia Baikousi. A Taxonomy of ETL Activities. In Proceedings of the ACM twelfth international workshop on Data warehousing and OLAP. In *ACM 978-1-60558-801-8/09/11*. DOLAP09, Hong Kong, China, 25–32. DOI:http://dx.doi.org/10.1145/1651291.1651297

Adam Warski. 2016. Kafka Streams: How does it fit the stream processing landscape? (2016). Retrieved July 2, 2016 from https://softwaremill.com/kafka-streams-how-does-it-fit-stream-landscape/

Fangjin Yang, Eric Tschetter, Xavier Leaute, Nelson Ray, Gian Merlino, and Deep Ganguli. 2003. Druid: A Real-time Analytical Data Store. In *ACM 978-1-4503-2376-5/14/06 SIGMOD'14, June 2227, 2014, Snowbird, UT, USA*. ACM Press, New York, NY, 12. http://static.druid.io/docs/druid.pdf

Online Appendix to:
PEACH Streams: Defining an OpenEHR Service Implementation with Stream processing for Computational Healthcare Cases in Radiology

ANDRE ALLORERUNG, University College London
ABHIMANYU CHAKRABARTY, University College London
CONNOR CLOONEY, University College London
NARMIN HUSEYNLI, University College London
EFTHYMIA KAZAKOU, University College London
DEAN MOHAMEDALLY, Univesity College London, Supervisor
NAVIN RAMACHANDRAN, University College London Hospital, Stakeholder

A. PEACH SUB-APPLICATIONS AND TEAM MANAGEMENT

The PEACH team consists of twenty people, five Software Systems Engineering students and fifteen Computer Science students. The SSE students were responsible for building the main PEACH infrastructure whereas the rest of the students were assigned into teams to develop four sub-application domains that were plugged into the main system. Each of the four sub-application domains of PEACH are presented in sections A.1-A.4. An overall idea of how the all applications are connected to PEACH system is provided in Figure 5.

In order to maximize the group's dynamic, we needed to adopt the Agile methodology principles [Collins]. We utilised Scrum methodology as the foundation of Agile project management [Salo and Abrahamsson 2007]. The team leader of the whole PEACH team was Ms. Efthymia Kazakou who also played the role of the Scrum Master of the overall development team . The first thing we needed to do was to distribute the team members into different teams according to the different projects within PEACH. The second step was to assign second leaders from the SSE students for each sub-team. These were:

— Ms. Narmin Huseynli, team leader of Radiology application (4 CS students)
— Mr.Connor Clooney, team leader of MDT and Clinical Trials application (6 CS Students)
— Mr. Abhimanyu Chakrabarty, team leader of AR/VR application (1 CS student)
— Mr. Andre Allorerung, team leader of Chatbot application (3 CS students)

Our communication with the clients and mentors was a two-way communication method and was taking place in sprint planning meetings each week and also via other communication channels (e.g. via email exchange, Skype and Slack application). As our main project management tool we used Microsoft Visual Studio Online where we allocated five different boards, one for each subproject in order to track the team's' software development process [Cohn 2005]. Git also enabled us to integrate our work on a single code base.

In order to effectively manage the teams, achieve our goals and keep track of our work we scheduled a series of meetings every week. These meetings consisted of three different categories:

(1) *Client meetings* which were scheduled for the architects team to get feedback and advice regarding the proposed architecture and team management.

 1539-9087/2016/09-ARTGS99 $15.00
DOI: 0000001.0000001

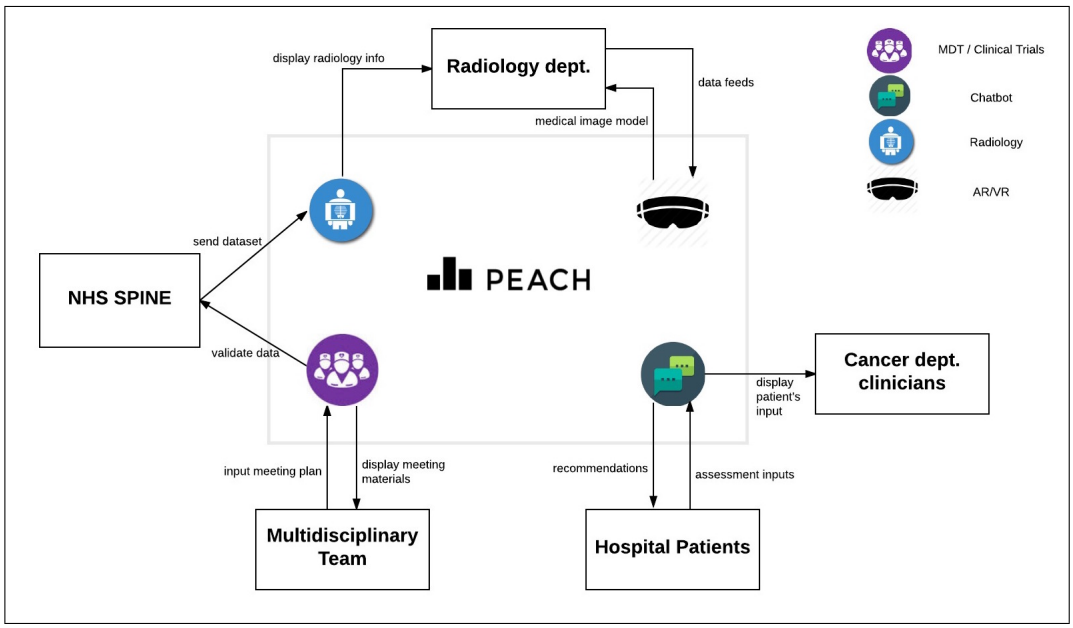

Fig. 3. PEACH Infrastructure Context Diagram

(2) *Weekly stand up meetings* where the entire Peach team came together to report their progress update and address challenges they faced.
(3) *Sub-team meetings* to review the work of each sub team.

A.1. Augmented Reality / Virtual Reality (AR/VR) application domain

This application is related to 3D organ modelling according to user specifications and it's intended to be used as a medical tool where a doctor can view exactly which area of the organ is affected and what can be the effects of the cysts. It is also considered as a surgical tool that can be used by surgeons to see the exact anatomy of a patient and estimate the complexity of the surgery beforehand. Our primary aim was also to project medical images and scans virtually which would allow the surgeon to interact with a 3D model of the scan. The team's current scope includes working with kidneys for the first iteration of this project.

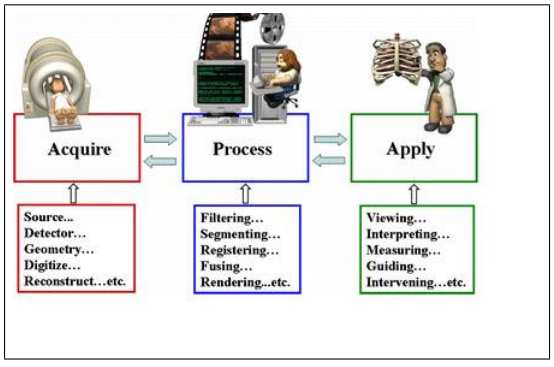

Fig. 4. PEACH AR/VR Flow of the procedure

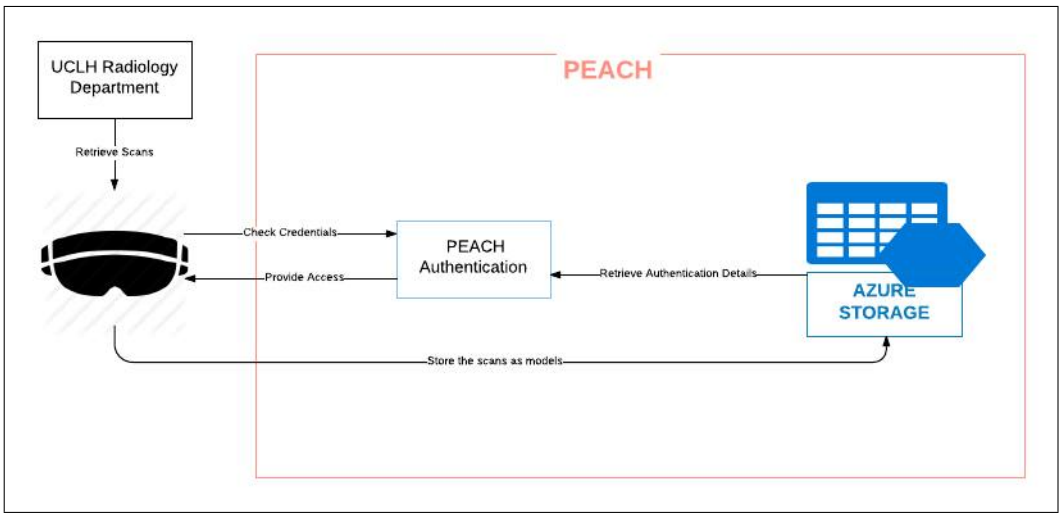

Fig. 5. PEACH AR/VR Architecture Diagram

A.2. MDT - Clinical Trials application domain

A.2.1. MDT. An multidisciplinary team (MDT) is a group of doctors and medical practitioners of various specialisations. This team would meet on several occasions in a month to review the progress of numerous patients and decide upon the next treatment or action that should be carried for a patient. MDT meetings are currently scheduled and managed using old and convoluted software. The aim of the MDT application is to provide healthcare professionals with a complete and consistent management application for an MDT meeting co-ordinator. Where they can manage all the patients that will be reviewed during the meeting, as well as the different healthcare professionals that are required to attend based on the patients being reviewed.

A.2.2. Clinical Trials. Throughout the NHS there are a variety of different regions that provide similar services but that are disconnected from each other, this is very apparent within the clinical trials area. The clinical trials application is intended to provide a connected clinical trials service for UCLH that could be expanded into different regions. Providing clinical trials search and filtering functionality to prospective trial participants and trial management functions to trial managers and administrators.

A.3. Radiology application domain

A.3.1. Visual Report. This is an application for reporting prostate MRI results in a graphical representation along with other cancer assessment and demographic details attached to patient records. The intention beyond that is an illustrative presentation of problematic areas by coloring them with different colors. It also allows doctors to add free text notes with regard to medical examination using a digital pen and performs handwriting and speech recognition techniques to convert ink strokes and speech input into a text.

A.3.2. Dashboard. The Radiology Dashboard application provides a personalised dashboard and reporting tool for multiple users for interactive analysis on event data that will take a direct data feed in order to provide real time statistics. Considering that each staff member could have different roles, users have access to a variety of services based on privileges given to them. Users are also able to customise their web page

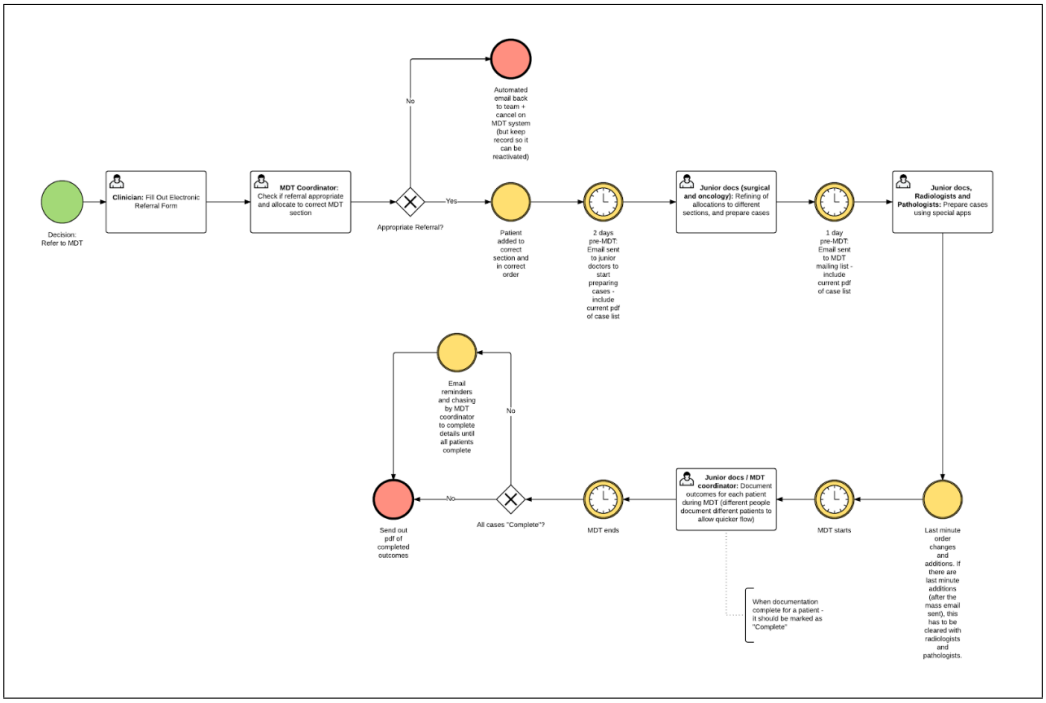

Fig. 6. Flow diagram of the MDT application.

that is role-specific by default. In the long term it's considered that the dashboard will aggregate much more details from patient records.

A.3.3. PEACH Generator. While leveraging patient sensitive data for a research study or further treatment purposes, preserving patient confidentiality and security is likely to be breached. The intuition behind the tool is to provide researchers and clinicians with anonymised patient records that give flexibility to them in terms of usage of the health data without concerning patient consent. The system generates novel patient datasets with low re-identification and anonymised data sets containing some utility for the purpose of data analysis. The tool satisfies the aforementioned requirements by utilizing perturbative (noise injection and data swapping) and non-perturbative methods (data shuffling and generalisation/ k-anonymity).

A.4. Chatbot application domain

The chatbot domain looks to develop an interactive conversational Chatbot using Artificial Intelligence (AI) and Natural Language Processing (NLP) techniques. The data captured can be used for updates to patient records. The aim of the chatbot is its ability to be plugged into other applications such as determining clinical trials for patients.

A.4.1. Component Architecture. From the high-level architecture created for the application, our team specified a more detailed image to the component-level diagram to define how the components interact with each other, within their own package or with components from different packages. This diagram is shown in Figure 10.

The following gives short description of each component:

(1) The chatbot web package, consists of four components:
    (a) Web graphical interface, UI facing design for users.

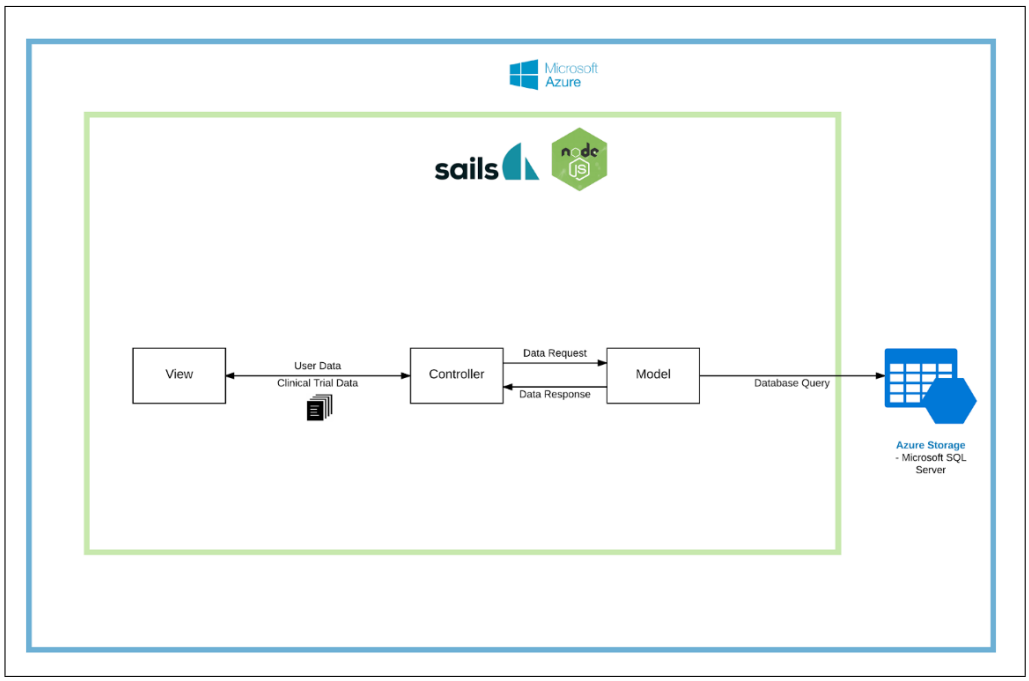

Fig. 7. Design of the Clinical Trials Application.

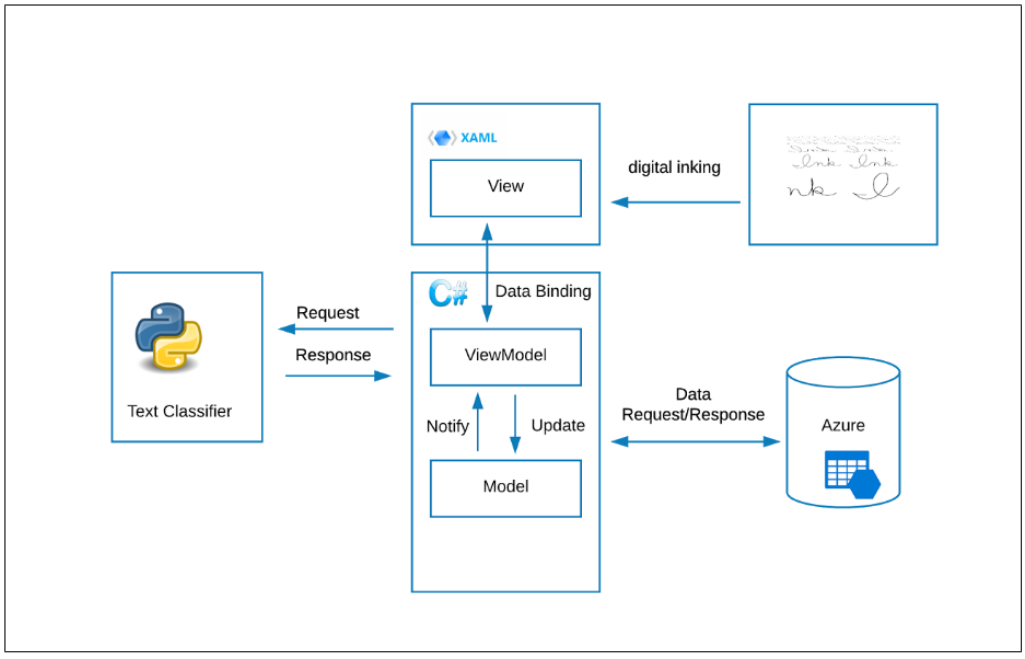

Fig. 8. Design of the Visual Report Application.

(b) Input capturer, interfacing directly the graphical interface. It consumes user's input via Flask library environment.

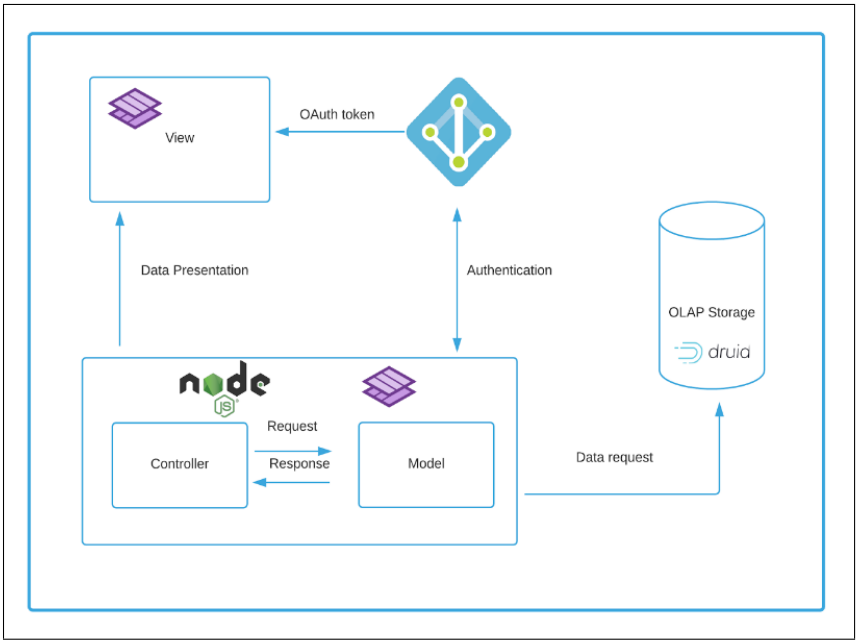

Fig. 9. Design of the Radiology Dashboard Application.

(c) Database connector, implemented using pyodbc library, incorporated with Flask.
(d) RESTful connector, handles communication to the majority of the application via RESTful actions.

(2) The chatbot brain, consists of four components:
(a) Categoriser, acts as the gate which retrieves inputs to be parsed to the brain.
(b) Preprocessor, handles pre-processing service for incoming input that has been categorised (e.g. stemming, etc.). It also handles the decision whether the incoming input will be processed as a concern type or a search type.
(c) Chatbot brain, generally the reply machine for the purpose of the patient assessment.
(d) Postprocessor, handles the pre-processing service for the reply generated by the brain (i.e. decorating the reply to the user).

(3) The chatbot recommendation engine, consists of four components:
(a) Searcher, basically the listener of the incoming queries from user or from the chatbot brain package.
(b) Crawler, handles the browsing feature for the search engine.
(c) Indexer, handles indexing of the list of websites that have been browsed using the crawler.
(d) Ranker, uses the PageRank algorithm for the website ranking system.

A.4.2. Requirements Documentation. As a tool to help validating requirements listed while developing the system, a goal model was created. This model depicts the roadmap, connections, and dependencies of each requirement, as shown in Figure 11. Listed in Table III are the requirements of PEACH Chatbot application which are derived from user stories and/or scenarios acquired in the phase. Each requirement is described as goals in KAOS Patterns and represents a node in the goal model.

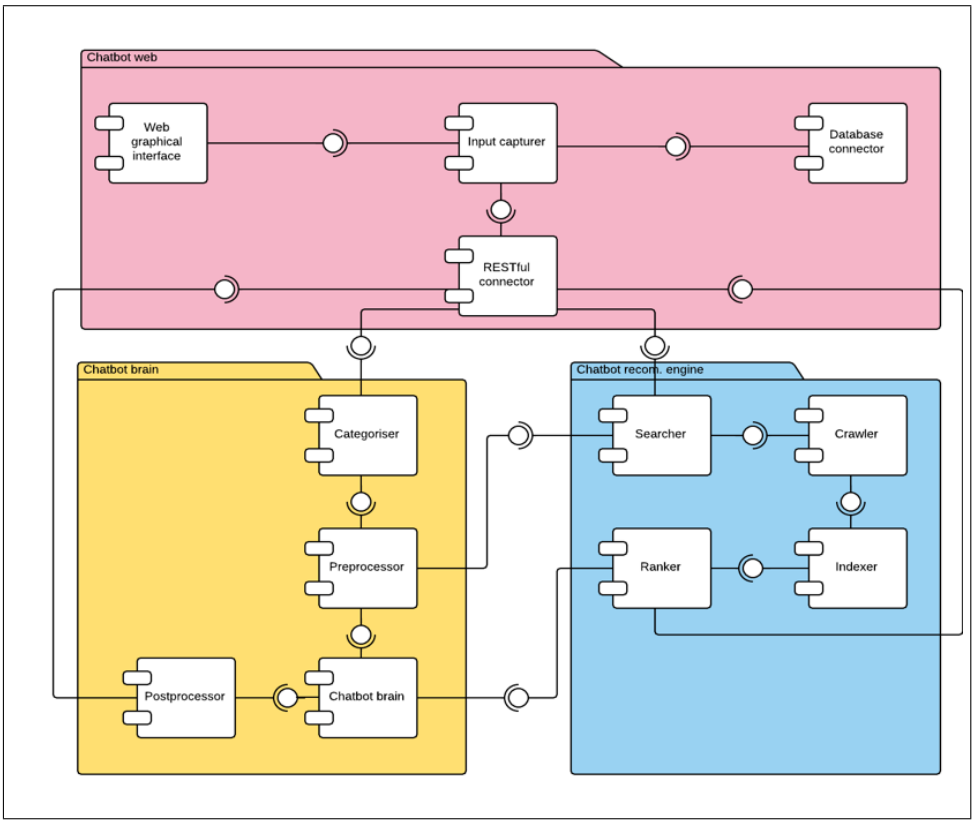

Fig. 10. PEACH Chatbot Component Diagram

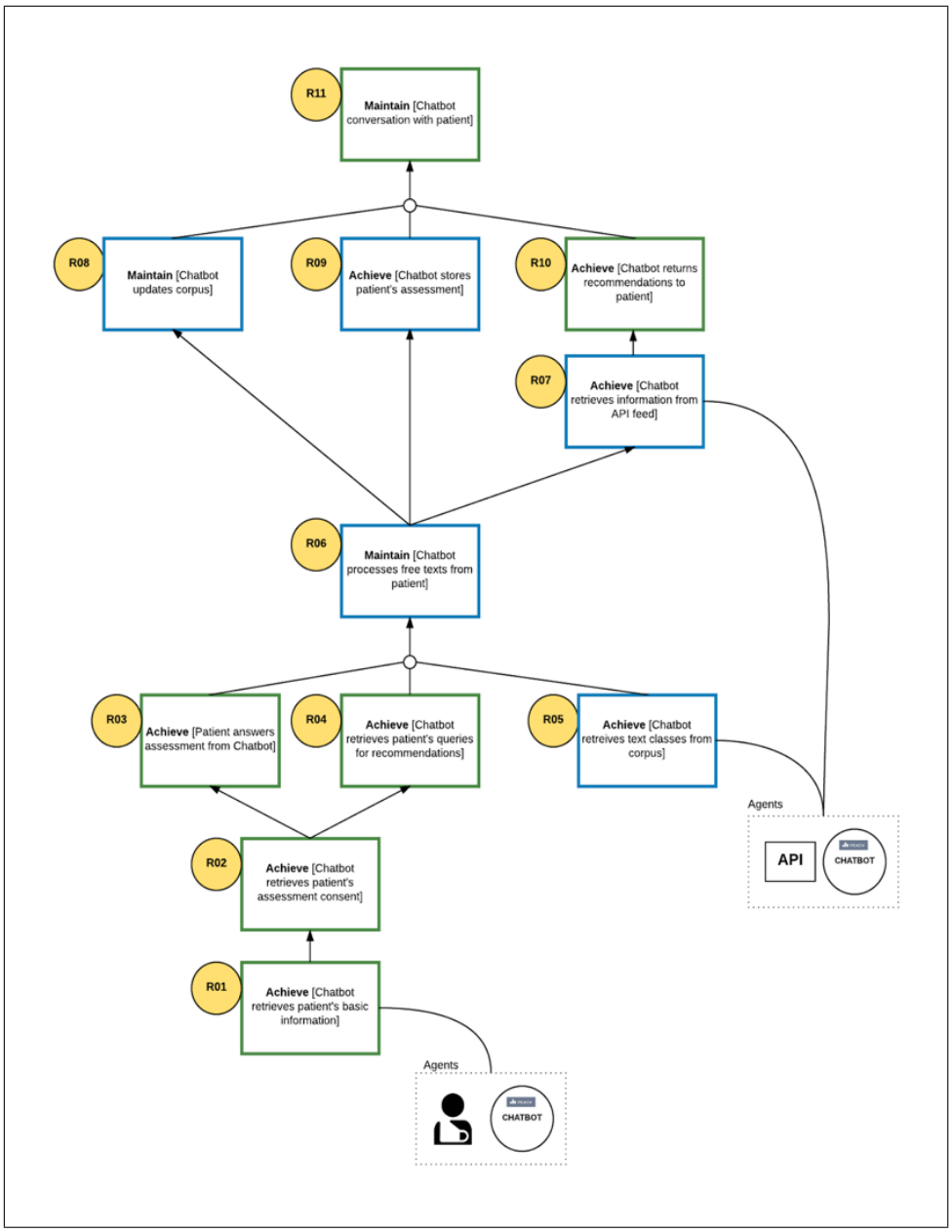

Fig. 11. Goal Model of PEACH Chatbot

Table III. Goals for PEACH Chatbot in KAOS Patterns

No	Requirements
R01	**Achieve** [Chatbot retrieves patient's basic information] **Desired Behaviour** **GIVEN** the patient is accessing chatbot app on his device **WHEN** the patient is instructed to input his name, date of birth, and PIN **THEN** chatbot stores the basic information for a session **AND** the application goes to the next section
R02	**Achieve** [Chatbot retrieves patient's assessment consent] **Desired Behaviour** **GIVEN** the patient has finished entering his basic information **WHEN** the patient is asked for his consent to use the application **THEN** the patient gives his consent to the system
R03	**Achieve** [Patient answers assessment from chatbot] **Desired Behaviour** **GIVEN** the patient has given his consent to the system **WHEN** the patient is in conversation with chatbot **AND** chatbot gives questions to the patient for assessment purpose **THEN** the patient will answer chatbot in free text format
R04	**Achieve** [Chatbot retrieves patient's queries for recommendations] **Desired Behaviour** **GIVEN** the patient has given his consent to the system **WHEN** the patient is in conversation with chatbot **THEN** the patient can query or ask question to chatbot in free text format
R05	**Achieve** [Chatbot retrieves text classes from corpus] **Desired Behaviour** **GIVEN** incoming inputs from the patient **WHEN** chatbot is going into text processing using NLP techniques **THEN** chatbot uses existing libraries/corpus for text dictionary
R06	**Maintain** [Chatbot processes free texts from patient] **Desired Behaviour** **WHILE** chatbot retrieves inputs from patient in free text format **MAINTAIN** chatbot processes the inputs using NLP techniques **AND** transforms the inputs into desired format
R07	**Achieve** [Chatbot retrieves information from API feed] **Desired Behaviour** **GIVEN** a free text format query or question from the patient **WHEN** the query or question has been transformed to structured format **THEN** chatbot system makes a request to external system for specific query **AND** chatbot retrieves desired information via the API
R08	**Maintain** [Chatbot updates corpus] **Desired Behaviour** **WHILE** chatbot is processing the free-text inputs using NLP techniques **MAINTAIN** chatbot updates its current dictionary
R09	**Achieve** [Chatbot stores patient's assessment] **Desired Behaviour** **GIVEN** the inputs from the patient has been transformed to desired format **WHEN** the patient has answered all the questions **THEN** the assessment inputs are stored into the data storage
R10	**Achieve** [Chatbot returns recommendations to patient] **Desired Behaviour** **GIVEN** the query or question from the patient **WHEN** chatbot has retrieved the desired information from external systems **THEN** chatbot displays a recommendation to the patient app
R11	**Maintain** [Chatbot conversation with patient] **Desired Behaviour** **WHILE** the session of the patient is still active for the patient app **MAINTAIN** chatbot and patient communication in form of conversation