The PACE System: A P2P Architecture for Cloud based EHealth Systems

Neil Donnelly
Bachelor of Science in Computer Applications

A thesis submitted in fulfilment of the requirements for the degree of Master of Science

to the

Dublin City University
School of Computing

Supervisor: Dr. Mark Roantree

October 2014

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Masters of Science is entirely my own work, that I have exercised reasonable care to ensure that the work is original, and does not to the best of my knowledge breach any laws of copyright, and has not been taken from the work of others save and to the extent that such work has been cited and acknowledged within the text of my work.

Signed: _____

Student ID: 13212223

Date: 09/10/2014

Acknowledgements

I would first like to take this opportunity to thank my supervisor, Dr. Mark Roantree, whose hard-earned expertise, willing advice and generous support made this thesis possible.

I would next like to thank my colleagues, Jim, Michael and Noel, for the collaborative and supportive environment you all contribute to.

I would also like to thanks all the people working on the dementia ELEVATOR project in the nursing building, Kate, Kirsty, Sophie, Paulina and Aoife.

To my family who I owe so much, of which, this thanks constitutes the bare minimum. I would like to express my extreme gratefulness and appreciation for everything you have given me and for supporting me whole-heartedly in completing this thesis. To Mam, Dad, Eimear, Aidan, Ronan and Roz, Lily and Aaron, and Rex, thank you so much.

To my wonderful girlfriend, Christine: thank you so much for always being there to help and support me, for putting up with me during the long days and for keeping me happy every day I was with you.

Contents

List of Figures

List of Tables

Abstract

Many healthcare facilities are currently investigating the possibility of utilising cloud computing platforms to store and share confidential patient records. However, before such a cloud-based system can be realised, there are still concerns over the safety of a patient's identity while their medical information is stored with a third party. Thus, the challenge is to provide a highly accessible system as provided by the cloud with security for patient data as required by legislation. In this dissertation, we propose a novel approach to protecting a patient's confidential information while still exploiting the benefits of the cloud platform. By extracting the identifying values from a patient's records and storing them on the machines of the healthcare professionals, the remainder of the anonymised record can be stored on the cloud. These two subsets of data can then be recombined into the original medical record when required by a user. This dissertation presents the PACE system architecture which combines modern cloud and peer-to-peer technologies to manage healthcare records on the cloud and enable the sharing of confidential information between users. The anonymised data kept on the cloud is available to all users while the sensitive local data which can identify the records are stored and shared across a decentralised hybrid P2P network formed by the clinicians.

Chapter 1

Introduction

As the world of technology has progressed, it has also helped in the advancement of healthcare. Technology has been instrumental in the improvement in fields such as diagnosis, medicine, treatment and patient monitoring[51],[14],[13]. The focus of this thesis is the improvement of patient record management and sharing between clinics. Securely collecting, maintaining and sharing patient records between hospital servers can be both arduous and costly [30]. Over recent years, healthcare operations in countries around the world have been investigating the possibility of migrating some of their medical systems to the cloud [1]. Cloud computing is the practice of using a network of resources to store, process and manage data on the internet rather than a local server. Cloud computing offers the possibility of offloading some of the burden of maintaining this data by storing records on the cloud while keeping it easily reachable and safely protected.

One of the many areas in medical research that could benefit from the improvements to data sharing is in dementia. Dementia is a serious loss of cognitive ability beyond what might be expected through normal ageing. The number of people currently estimated to be suffering from Dementia is 44 million worldwide, and is expected to rise to 76 million by 2030 and 135 million by 2050 [48]. These staggering numbers emphasise the social and economic cost of the disease and the need to dedicate resources to reduce these swelling numbers. While dementia is a chronic progressive illness with

no cure, there is now strong evidence that the effects of dementia can be allayed by adopting lifestyle changes in mid-life that aim to improve cardiovascular health, low mood, and diets and increasing physical and cognitive activity[32][29]. Increasingly, the importance of dementia prevention and the need to take preventative measures based on existing knowledge are being highlighted at an international level. Some patients with dementia are unable to find transport to a clinic [16] and so keeping healthcare records available on the cloud would be extremely helpful. Clinicians can instead meet patients in their own homes where they feel most comfortable and safe and still remain connected to all the data available to them at the clinic. This access to data wherever the clinician is, can help with both the diagnosis and treatment of patients [9].

The research presented in this thesis has been conducted as part of the ELEVATOR project [18]. The project is a collaboration between experts in the field of dementia seeking to build a new cloud-based data management system for dementia patients. ELEVATOR is funded by the HSE (Irish Healthcare system) and Atlantic Philanthropies. The aim of the project is to examine the current deficiencies in the care of people with dementia within the community and to identify eight areas where education and training can improve the lives of people with dementia and their carers. These eight areas are: conversation; non-verbal-communication; environmental consideration; anxiety reduction; mindfulness and empathy; understanding behaviours, retaining a sense of self; checking and understanding [28]. ELEVATOR has developed an Educational Needs Report which maps the education and training needs for stakeholders. As this education involves general practitioners, community nurses, other healthcare professionals, carers and community groups, there is a requirement for an infrastructure that facilitates cooperation across the different healthcare units. This thesis is aimed at meeting these requirements. By combining cloud and peer-to-peer technologies, an application could be developed that facilitates the need for the availability and sharing of patient records between these groups of healthcare providers.

The Introduction chapter is divided into the following sections: section 1.1 and section 1.2, cloud computing and P2P are described and their place in the world of healthcare is explored. In section 1.3 the challenges of migrating healthcare records to the cloud are explored. Section 1.4 sets out the goals and objectives for this research. Section 1.5 presents the conclusions to be drawn and describes the structure of the thesis.

1.1 Cloud Computing

At its core, cloud computing has existed since the early days of the internet as it is fundamentally a delivery of resources and services delivered over the internet[53]. However, it is only recently the term 'cloud computing' has gained traction and become one the most pervasive terms in the world of technology. As internet technologies have advanced so has the speed and efficiency of cloud computing[20]. The appeal of having data and services available at any time has captured the interest of both businesses and consumers. The American National Institute for Standards and Technologies (NIST) has defined cloud computing as has having five essential characteristics [42]. *On-demand self service* means any authorised user can utilise the service without the need for human interaction from the service provider. *Broad network access* asserts that the service is available over the network and is accessible through any standard internet device (PC,tablet, phone, etc.). *Resource pooling* is the pooling of a services resources and the dynamic assigning and reassigning of these resources to serve the needs of a user. *Rapid Elasticity* means the amount of resources at the disposal of the service can be increased and decreased (sometimes automatically) to meet the varying demands of users. The final characteristic is *measured service*, which describes cloud systems having the capacity to measure its resources and the demand for those resources to control and optimise the delivery of its services.

These five characteristics combine to form a powerful and flexible tool which provides the scope to make patient records accessible at any time, from any location with an

internet connection and with any WiFi enabled device. Up-to-date patient records can also be easily shared between users from different clinics or hospitals. Smaller healthcare facilities, such as medical practices and laboratories who may not have the resources to hire internal IT staff, could maintain their own patient records or even access records of other larger facilities via the cloud[30]. There are also the inherent non administrative benefits of making use of the cloud. There are virtually no limits to the amount of data that can be stored as storage is dynamically added and removed to suit the needs of the customer.

Cloud computing's place in the healthcare industry, however, is dependent on whether the trust of the lawmakers and practitioners can be gained for the safety of the cloud for confidential information. As often happens with emerging technologies, the law cannot keep pace with the rate of progress which inhibits how useful these technologies and techniques can be. Although trust in cloud computing is growing [43], there still exists a hesitancy to adopt cloud computing for storing something as sensitive as patient records, even though most cloud providers provide assurances of both data redundancy and protection [41] [2] [25]. Until such a time that trust is absolute, private patient information cannot be stored on the cloud in many countries. This makes sharing patient records between clinicians very difficult as the identifying features for the data cannot be accessed from the cloud, severely diminishing the benefits of using such a solution. This key issue must be addressed in any cloud-based solution for healthcare and is the focus of this thesis.

1.2 Peer-to-Peer Computing

A peer-to-peer (P2P) network is a decentralised, distributed network architecture comprised of multiple nodes (peers) which both generate and distribute data. On a completely decentralised P2P network, every peer is considered equal and so responsibility for finding and sharing data is dependent on an algorithm shared by every peer. The peers are connected in a web of peers are connected to neighbours and know only about those neighbours not the whole network. Some P2P architectures

introduce the concept of a super-peer, which acts as an organiser for standard peers while still fulfilling the responsibilities of a normal peer; such a network is known as a hybrid decentralised architecture [4]. Peers are assigned super-peers which can communicate with other super-peers on their behalf and improve the performance of the search and delivery of data, data redundancy, and peer organisation [6]. P2P networks are also quite scalable; the system grows with each user dynamically. Data availability is also inherently improved as now there are multiple sources from which one can find information.

In the context of the healthcare industry, such a hybrid decentralised architecture has a number of benefits. For instance, sharing and accessing information between users is made very simple as each user is sending the information directly rather than over a server [5]. The patient records are also kept redundant as they are shared amongst several peers. As this is also self-organising, there is no need for a large server to maintain the network. The model itself is reflective of the real world model of clinicians and clinics. Each clinician is associated with a clinic and is aware of the clinics other clinicians, a clinic can communicate with other clinics on behalf of the clinician and information is automatically shared between clinicians of the same clinic. However, using a P2P network comes with some issues. As each peer represents a clinician, the responsibility of storage for patient records is distributed to each clinician. However, as the database of patient records grows, the size of data kept by each clinician will steadily increase so although the network is scalable for the number of users but there is a quite likely a varying limit on storage for each peer.

1.3 Problems and Motivation

For many in healthcare, the idea of electronic healthcare records offers a meaningful step forward in the treatment of patients. It offers a means for records to be made accessible at all times whenever needed. This ensures that the clinician has the most up to date version of the patients records wherever they are. Having an updated

records improves the clinicians understanding of the condition of the patient, leading to a more informed diagnosis and treatment. These records can also be transferred between authorised personnel quickly and easily. To provide the accessibility needed for these records to fulfil this design, the cloud seems the obvious place to host these records due to the low cost and availability[52]. The challenge for migrating this data to the cloud however, comes from the issues with trusting a third party cloud provider. In the current scenario, each clinician is entrusted with the personal details of their patients and so they are unwilling to share or keep these details in places vulnerable to intrusion. So, how can a healthcare provider take advantage of the cloud to share their records with others while still managing the issue of privacy?

The focus of this research is the sharing of patient records on the cloud between users while still respecting the confidentiality of a patients identity. Without the ability to share a patients full record, including their identity, the true usefulness of migrating this data to the cloud is made redundant. However, this must be done without breaching the privacy of the patient. A means of connecting users in a network to expedite the sharing of data on the cloud would be ideal. In order to accomplish this aim, there should be a focus in modelling the real world infrastructure of clinicians and clinics. By modelling a large organisation of clinics and clinicians, the place of the user within the system and the patient records available to each, the devised system should facilitate both the searching and retrieval of patient records from others on the network. No such system currently exists on a scale large enough to sustain the participation of multiple healthcare facilities and clinics and thus merit the use of cloud technologies.

1.4 Hypothesis

For healthcare providers, the lack of trust in the cloud to keep private information confidential is a major issue that is slowing down the technological progression in the industry. As not all information can be stored on the cloud, sharing information between healthcare workers presents a serious barrier to adoption. In order to move

forward, a means of sharing data between clinicians using a cloud-based system must be developed. This research hypothesises a potential solution that allows the bulk of patient information to be kept on the cloud while also allowing entire records to be shared. Our hypothesis is to divide the patient records into two types, fields that can be used to identify a patient and fields that are purely medical related that could not identify a patient. The non-identifying medical data makes up the bulk of the record and is stored on the cloud to take advantage of the storage capabilities. The identifying information is then kept by the clinician on their machine and shared directly with other users. The research questions that must be answered to justify such a system and the proposed solutions can be summarised as:

- By placing the users into a P2P network in conjunction with the cloud application, can the data be successfully sent between users without ever being transferred over a cloud server?
- Could the real world organisation of clinics and their clinicians aid in the modelling of a P2P network that facilitates the searching, sharing and redundancy of data?
- Could a simple query language, such as a pared-down SQL-like language, be used to search and retrieve data from various data sources and recombine the results into a complete table of results?
- Can such a system that involves the division and combination of data present a viable solution to sharing patient records stored on the cloud without endangering a patients identity or will the performance costs of such mechanisms prove too costly to the applications usability?

By detaching the identifying data from the records being stored on the cloud and keeping it stored between the users of the system, we should be able to take advantage of the storage and mobility of the cloud. By using P2P technologies to share the identifying information between these users directly, we preserve the sharing ability

required for the system to operate as it should without sacrificing the performance of the system.

1.5 Conclusions

In this chapter, the problem of patient record management across healthcare facilities was introduced. The use of cloud computing as a possible solution to both offload the IT related costs associated with patient record maintenance and the potential of sharing patient records between healthcare workers via the cloud. As discussed in this chapter however, there is a concern over the safety of sensitive patient records being stored on third party servers. The hypothesis of this research proposed dividing the data and keeping all data that could identify a patient on a users machine, then sharing the sensitive patient values directly with other users. This allows for the benefits of the cloud to be exploited while ensuring entire patient records can still be safely shared.

The remainder of this thesis is structured as follows: Chapter 2 describes the current state of research into migrating health records to the cloud and the place P2P can play in healthcare. The chapter identifies some of the strengths and weaknesses of such projects in relation to the objectives of the research set out in this thesis. Chapter 3 introduces the PACE architecture and the components that make up the model, the part each component plays and how they interact with each other. Chapter 4 details the query language developed for use with the PACE architecture including what functionality it must accomplish, how the query is parsed and transformed. In Chapter 5, the experiments used to evaluate the PACE design and architecture are detailed, including what functionality was implemented and how it performed in a real world context. Chapter 6 presents the conclusions that can be drawn from the research and proposed areas for further research.

Chapter 2

Related Research

In the previous chapter, we motivated the need for a cloud-based EHealth system that allows registered users to upload and share patient records while keeping confidential patient values safe. Our hypothesis focused on using a hybrid P2P architecture in conjunction with the cloud to keep personal information local and still allow users to share data. The hypothesis is in response to reluctance among the healthcare industry to adopt cloud technologies due to a fear of patient data being vulnerable on the cloud. Over recent years, there has been an emergence of solutions seeking to solve this same problem. There have been solutions aimed at providing greater levels of security to data stored on the cloud, others have attempted to use other technologies outside the cloud to achieve the same functionality and some have attempted to build upon the standard cloud architecture.

In this chapter, we will examine some of these state-of-the-art solutions and how they influenced our own research. When analysing these projects, we will focus on their scale and their ability to allow users to share and find data on the system. The chapter is structured as follows: section 2.1 focuses on projects relating to cloud-based solutions to safely sharing patient data, section 2.2 examines a P2P centred approaches to the problem and section 2.3 describes some projects that seek to solve the problem using expanded cloud solutions. In section 2.4, we will present our conclusions.

2.1 Sharing Confidential Data on the Cloud

In regards to confidential data on the cloud, in current research, the goal for protecting this information has been mainly focused on encryption and introducing domains to segment the data [39]. All the data is encrypted before being uploaded and decrypted by authorised users on download. However, this solution offers a new set of challenges: how can data be shared effectively between multiple types of users? Could all users be given the same decryption key? However, this makes all the values of a patient's records available to all users which may not be a desirable outcome for the patient. Companies such as Microsoft and Google have both attempted to allow patients to have a measure of autonomy over their own health records and control over who is capable of seeing these records [55]. However, the main issue is providing a system which allows healthcare workers to share patient records without exposing sensitive information to other users of the system.

2.1.1 Overview for Attribute-Based Encryption for Patient Records

One suggested solution that allows patients to share information between different healthcare entities is the use of multiple decryption keys and attribute-based encryption [26] to protect different domains of patient data. Such a solution was presented in [35] and then detailed further by the authors in [36]. The proposed solution sought to allow users to decide what other users could see from their personal health records. The paper takes a patient-centric approach to personal health records (PHR), meaning the patient creates their own record and decides who can see it. The patient also creates and distributes the decryption keys to other authorised users such as nurses, friends or pharmacists. The health data is then classified according to certain attributes, for instance, whether the information is relative to an illness or an injury and what healthcare facilities were visited to treat the issue. These attributes are then used to generate keys that encrypt the data from users that do not have

decryption keys matching those of the attributes. A similar approach is presented in [11], which introduces a framework based on Cloud-based Privacy-aware Role Based Access Control (CPRBAC). The CPRBAC solution encrypts patient data, and relies on roles to decide the decrypting capability of users.

Figure 2.1 demonstrates a sample security policy put in place by a patient. Briefly, this is interpreted as a user must be from either *Hospital A* or *Hospital B* and either a *Physician* with an *M.D.* working for *Internal Medicine* department or a *Nurse* in the *Gerontology Nursing* department.

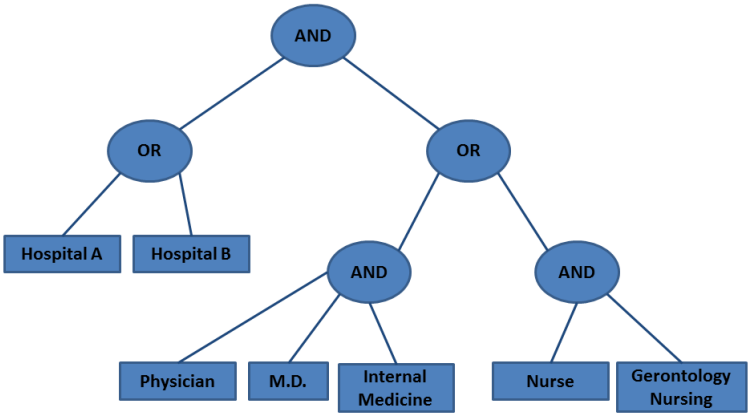

FIGURE 2.1: Model-View-Presenter Design

To allow other users access to the data without the need for outright permission form the patient, the authors also divide the system users into two security domains: the *Public Domain* (PUD) and the *Personal Domain* (PSD). Each user of the system has control over their own PSD and should be comprised of users that they have had personal contact with. The PUD contains professional users from various sectors of society relevant to healthcare, such as a research group from a university or an insurance company. Each professional user is given a set of attributes with which they can access certain information on multiple users PHRs. To ensure that users that are part of the PUD are able to see only what is relevant to their sector, several *attribute authorities* are used to govern a set of attributes for the PHRs from which the users can acquire the requisite keys for a user, without needing to gain direct permission.

The same authors, in another paper [34], present a means of querying the data stored as part of the attribute-based encryption using hierarchical predictive encryption [46]. Their scheme, APKS (Authorised Private Keyword Searches), allows for multi-dimensional range queries while maintaining query privacy simultaneously. This is done through the use of *local trusted authorities* (LTAs), which are charged with determining a users search privileges. The LTA accomplishes this by isolating the underlying attributes necessary for the query and then checking to ensure the user has the privileges to search for those attributes. The LTA should also be able to associate restrictions on a query based on the users attributes. For instance, if a doctor from *Hospital A* searches for all patients with *diabetes*, the LTA should first check that the doctor has the privileges associated with *diabetes* and then restricts the query to only check patients in *Hospital A*. As all the data is encrypted and stored on the cloud, HPE is necessary to predict keywords in data without leaving the data vulnerable to attack. The query is converted into a compatible form, encrypted with a security key corresponding to the data and then used to retrieve encrypted data. The results of the query can then be decrypted and returned to the user.

2.1.2 Limitations

Although attribute-based encryption is a solid fine-grained approach to storing patient data on the cloud, and can be manipulated to allow the data to be queried, we believe the approach developed through our research provides a more agile and inherently safe system. There are always performance costs associated with encryption and decryption, and judging by the work presented in [36], these costs are incurred with almost every operation on the data. As the data is encrypted, querying the stored data is made quite difficult and although a solution is presented, it requires effort in order to structure and store the data in a format capable of being searched and still requires cryptographic operations to function. Although the use of LTAs for each group of users to enforce privileges on data access is a better approach than through a central server, they also appear to be a large draw on resources, as each group requires a machine for each group of users.

2.2 P2P Solutions for Sharing Data

Sharing patient data between healthcare facilities quickly and securely is a key goal for the modern healthcare industry. Ensuring that healthcare workers always have the most up to date information on a patient is vital in the diagnosis and treatment of patients. P2P technologies appear to be a logical solution to the issue [17]. Information can be directly shared between users without the need for information to be wholly stored at a central repository. In this section, various solutions to sharing patient information between users in a healthcare environment will be provided.

2.2.1 P2P Technologies in a Healthcare Environment

In [38], the authors present a P2P based strategy for sharing patient information between users in a healthcare facility. Patient data generated by *body area networks* (BANs) are shared between clinicians using JXTA [21], a java based open source protocol for P2P communications. The system described in the paper is designed as a hospital spanning solution to make patient data generated by body sensors available to all healthcare workers employed by the hospital, regardless of location. The hospital workers acts as peers on the system and are organised into peer groups, with each group representing a different medical department. Patient information can then be shared between all users of the same peer group, under a P2P environment. A JXTA *Relay* is used as a means of communicating between peer groups and sharing data from one group to another. Figure 2.2 demonstrates the architecture proposed by the authors in [37]. By using P2P technologies, the patient information can be shared between users dynamically with a degree of fault tolerance. Dividing the peers into groups also ensures that patient data is not shared with users that do not require the information.

The previous solution presents a means of sharing data *within* a hospital however, it would be more beneficial if multiple healthcare facilities could communicate on the

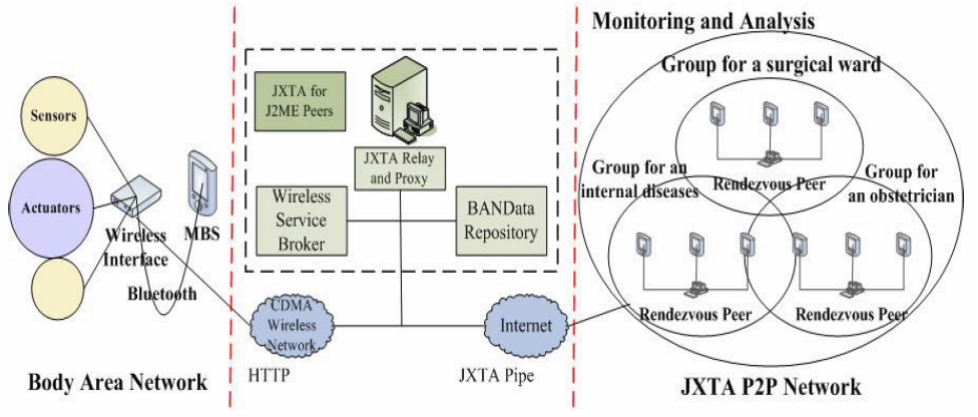

FIGURE 2.2: System Architecture Presented in [37]

same network. Such a network is presented in [10], with a solution for sharing meta-EPRs (electronic patient records) between operators working in different hospitals using a hybrid P2P architecture. In the described architecture, super-peers act as a central server for peers and connect to other super-peers, forming the network. The super-peer also manages an XML database of meta-EPRs, extracted from the hospitals stored data and records inserted by the users of the system. This is a similar solution as to the one presented in [27], which represents healthcare centres as peers. However, in [27], each healthcare facility is represented by a super-peer, and each user by a peer. This super-peer solution is a better representation of modern healthcare operations, allows workers to maintain the healthcare centres data and allows peers to make individual data requests. The architecture is demonstrated in figure 2.3.

The peers of the system contribute to the super-peer's database of meta-EHRs and in turn can search and retrieve data from across the network via the super-peers. The steps involved in retrieving data from the peer network are as follows:

1. The user specifies parameters for search through the application.
2. The peer submits a request to its super-peer.
3. The super-peer searches its local database using the specified parameters.
4. The query is then forwarded to the other super-peers who perform a similar search.

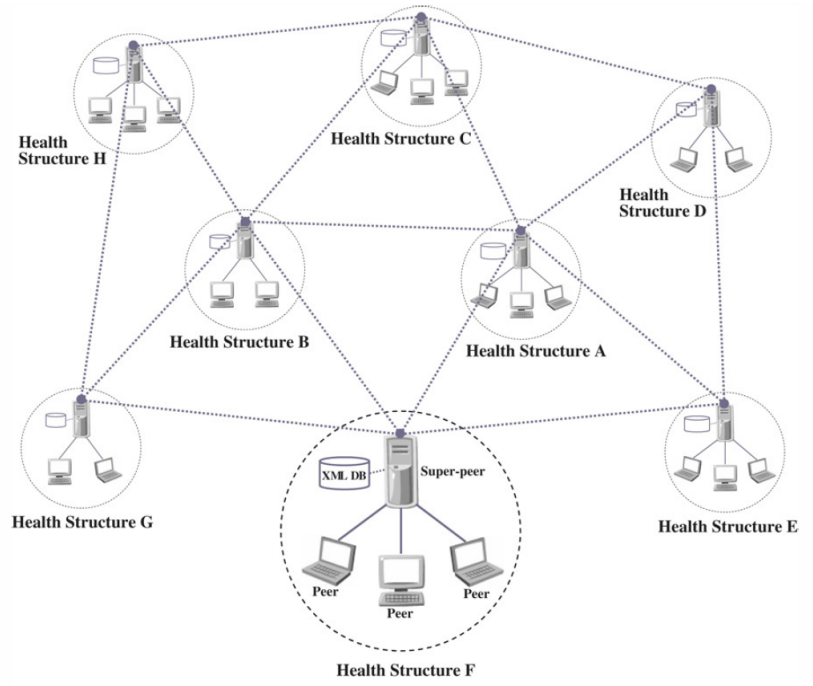

FIGURE 2.3: Peer Network Architecture Presented in [10]

5. The other super-peers return their results to the requesting super-peer.

6. The super-peer sends the results to the peer, to be presented to the user.

The use of super-peers and following the steps outlined above, the process of locating patient data spans the entire network with a relatively small amount of required connections.

2.2.2 Limitations

The research presented in [38] suggests using a P2P architecture to share a patient's health information collected from a BAN is very useful for sharing a large amount of information between various users and the use of peer groups allows subsets of users to share relevant information between each other. However, the bulk of the data still requires being stored on hospital grounds which can become expensive compared to using the cloud and restricts the solution from expanding to include other facilities.

[10] presents a P2P architecture designed for inter-clinic sharing. The system of super-peers allows for each healthcare worker to effectively query the network of peers and each clinic to manage their own users. This solution does produce a burden upon the super-peer, as it is responsible for storing and distributing all the patient data for the represented clinic. The super-peer must also communicate with the network for every query and retrieve and send data on the peers behalf. As the role of super-peer is performed by a server and not a standard peer with elevated capabilities, if the server fails an entire clinic disappears from the network until it is repaired. Thus, this architecture reduces the fault tolerance benefits of P2P networks.

2.3 Using the Cloud as a Platform

In section 2.1, we presented multiple cloud-based solutions for managing patient data and the limitations of those solutions. In this section, we will present research that aimed to expand upon the cloud platform using various technologies and techniques. The two additions made to the standard cloud architecture are the use of hybrid clouds and the introduction of P2P networks. Both of these solutions aim to provide a safe and more efficient means of retrieving and sharing patient information between users by attempting to have local resources perform some of the functionality usually left to the cloud. This cloud functionality could be storing certain portions of the data closer to the user to increase performance or providing security operations such as encryption and decryption.

2.3.1 Hybrid and Multi-Cloud Solutions

A hybrid cloud architecture is a combination of two or more cloud infrastructures, such as private and public clouds, that can communicate with each other and provide data interoperability [42]. In [12], the authors present a hybrid cloud solution for EHR management. The authors propose combining a public cloud with various hospital-specific private clouds to enable users across several institutes to share patient data.

All patient data is encrypted and stored on a hospitals private cloud and with the public healthcare provider. Access to the data is also protected by several layers of security, including a requirement for specialised key-cards. This ensures that all healthcare facilities have access to data and data relevant to a hospital could be retrieved quickly from the local cloud. Any clinics that do not have the ability to install a private cloud would be given access to the public cloud, which also acts as a safety measure in case of a failure on the private cloud. The infrastructure can be seen in figure 2.4. The infrastructure is similar to that presented in [40], which presents a domain structure for public and private clouds but is designed to provide an IaaS (Infrastructure as a Service) [7] platform for commercial use.

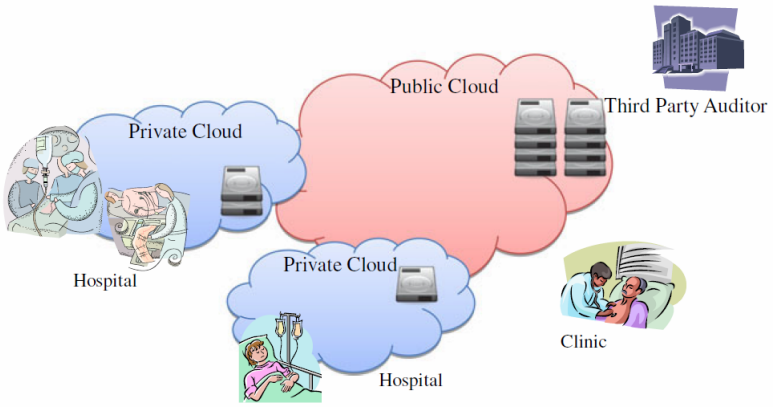

FIGURE 2.4: Cloud Infrastructure as presented in [12]

In order to retrieve data from the private cloud, the patients smart card must be used to acquire the decryption license from the public cloud. This license can then be used to find and decrypt the data stored on the private cloud. In [54], the roles for the public cloud and private cloud are reversed. The private cloud authorises users access to the data and provides the decryption keys in order to retrieve data kept on the cloud. Using the private cloud as an authorisation wall to the data kept on the public cloud means that the data does not have to be stored twice and can instead be kept solely on the public cloud.

The authors of [19] present a method of sharing healthcare information between users using multiple clouds. A patients data is divided and distributed across multiple providers to reduce the adverse effects of *curious* cloud providers. A *curious* cloud

provider is described as a provider that honestly protects and operates on data but also attempts to learn what it is storing. A *multi-cloud proxy* is used to encrypt, distribute and recombine patient data from multiple sources. Patient data is split using a secret sharing scheme and identifiers are generated for each cloud provider that can be recalculated by other authorised users. In order to retrieve the data, the proxy must collect the shares of data from each provider and combine the shares into a full record which can be presented to the user.

2.3.2 Combining the Cloud with P2P Technologies

By combining P2P technologies with the cloud platform, information can be shared directly between different users which can be preferable in certain circumstances. The authors in [49] present an integration of a JXTA P2P network with cloud computing to enable healthcare workers to gain quicker access to patient information in an emergency scenario. Patients and healthcare workers are peers on the system and provide information to be stored on *community clouds*. These *community clouds* are managed by a central cloud application, known as the *cloud controller*. For each peer, their information is stored on the cloud and connections are made to neighbours and relatives. Healthcare workers are represented as *proxied peers*. These peers can only join a network through certain peer relays and form a community of healthcare professionals who may be contacted with alerts for incoming patients. This infrastructure is demonstrated in figure 2.5. In the case of an emergency, ambulance crew are sent patient information from the cloud which can be augmented by medical histories from relatives or can be used to contact neighbours. From the ambulance, the crew can also alert hospital staff to the incoming patient and depending on the emergency can alert specific specialists via P2P communications. Thus, this infrastructure requires the need for multiple tiers of users and role definitions.

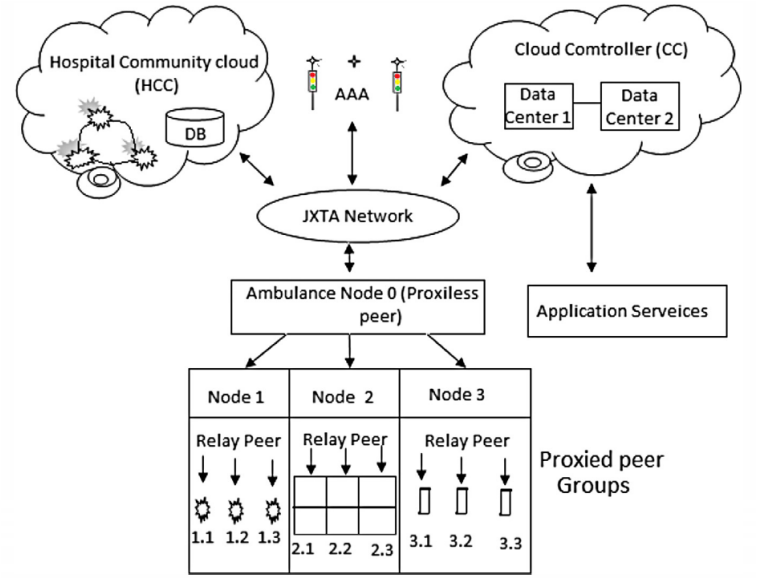

FIGURE 2.5: Cloud and P2P Infrastructure as presented in [49]

2.3.3 Limitations with Both Approaches

Although the hybrid cloud solution provided in [12] uses private clouds for hospitals as a means of allowing quick access to the patient data, its implementation in this instance appears redundant. All the data stored on the private cloud is stored on the public cloud so there are multiple copies and the benefits of using private clouds are diminished when the public cloud is used for authorisation so the user must interact with the cloud anyway. Using the private cloud to authorise access and the public cloud to store data, as presented in [54], is an improvement. However, if the private cloud is used solely for authorisation, then there is little benefit in implementing a private cloud as scalability becomes less of a requirement as no data needs to be stored.

By distributing data over multiple clouds, as presented in [19], a patient identity becomes much more difficult to discern by *curious* clouds or access violations. However, the use of multiple cloud providers to store data does not seem a cost effective approach for providing this level of protection. The level of heterogeneity between the different cloud providers could also become a serious problem to overcome. This distribution of data and heterogeneity would also make querying the patient data extremely difficult.

By using P2P networks, the authors of [49] have ensured that the various networks of users can scale with the cloud and facilitates direct communication of urgent patient data. However, the use of several clouds managed by a single managing cloud seems a waste of resources compared to having a single cloud which can expand to cater for the same number of users without a reduction in performance. The paper also suggests that all peers be self-organising into their own groups and must also facilitate queries. By introducing a super-peer structure to the network, peers could be more easily managed into groups and queries could be resolved much quicker, as presented in [57] and mention in section 2.2 with [10].

2.4 Conclusions and Final Analysis

In this chapter, we have examined multiple projects attempting to provide a method of safely storing and sharing patient data between healthcare clinicians. The projects examined were separated into three categories: solutions based solely on the cloud, architectures that supported sharing confidential information via P2P communications and finally solutions that expanded upon the cloud platform to provide efficient mechanisms for sharing and storing data for multiple clinics. The main benefits and limitations of each project were identified and used to inform the design and architecture of our own system, which uses a novel combination of cloud and P2P technologies. The main points learned from our analysis of each category are as follows:

Sharing Confidential Data on the Cloud

1. **Limits of Encryption**. Many solutions for managing confidential data on the cloud focus on encryption as the means of protecting the information. However, all encryption and decryption come with performance costs, so a solution that refocuses safety away from encryption could improve the performance. Encrypting data also makes querying very difficult and although a solution was

provided in [34], the queries are limited and require data to be manipulated into a format suitable for queries.

Sharing Patient Data over a P2P Network

1. **Organising Peers into Groups**. Although a P2P system is scalable and dynamic, considerations must be made towards how much data each peer should have and how easy data can be found on the network. Peer groups improves retrieval performance as in order to find a peer, one simply has to find the right group which is significantly quicker than finding a single peer on the network. Peer groups also allow data to be distributed across groups to reduce the amount of data needed to be stored by a single peer.

2. **Super-peers**. Super-peers appear to be a very good way of improving the flow of communication between the central server on the network and other peers [31]. As described in [10], super-peers form an interface between the network and a peer and can help distribute data to the peer and queries across the network. However, it is also important to allow peers to communicate directly with one another when sharing data to reduce the load on the super-peer.

Expanding on Standard Cloud Infrastructure

1. **Data Distribution**. By dividing the data and storing most of it on the cloud and distributing the identifying features across a network of users, we can help improve data redundancy for the identifying values and also help protect the integrity of patient identities from *curious* cloud providers or an attack on the provider.

At this point, the current state of managing patient data using cloud technologies has been explored with the current trends and failures discussed. The above projects propose strategies for safely sharing confidential data between authorised users and most involve leveraging the cloud as the central data server for this information.

However, each project has some limitations which make it unsuitable for solving the overall problem of making the most of the cloud while allowing users to safely send data between each other. Therefore, our solution focuses on sharing data between users while still taking advantage of the considerable storage and computational capacity of the cloud. In the following chapter, we outline the design and architecture of our proposed solution.

Chapter 3

PACE System

In Chapter 1, the hypothesis of this thesis was introduced which proposed that by separating any data that could be used to identify a patient and by storing the rest of the data on the cloud, one could take advantage of the benefits of the cloud and still share patient records safely between users. The identifying information would instead be stored on the user's machine and shared between the other users. One of the research questions this research is aiming to answer is whether P2P technologies could be leveraged to facilitate this sharing between users? Based on the analysis of current research presented in chapter 2, we identified the flaws in current solutions and developed a novel combination of P2P and cloud technologies to solve the problem. In this chapter, the PACE (P2P Architecture and Cloud-based EHealth) system is introduced [15]. The PACE system combines cloud and P2P technologies for the purpose of storing patient records and sharing them across a network of clinicians from multiple clinics.

The chapter is structured as follows: in section 3.1, the set of problems the PACE system was designed to tackle and a brief description of the principles behind its design are provided. The requirements of the system are expounded upon in section 3.2 and the design of the system is described in section 3.3. Section 3.4 outlines the architecture of the cloud portion of the PACE system while the components and processes of the client portion are explored in 3.5. Section 3.6 then describes the

technologies that are leveraged by the PACE system before the chapter is concluded in Section 3.7.

3.1 Overview

The PACE system's primary functionality is the ability to store and review patient records on the cloud. However the real challenge for the system is focused on sharing these records between clinicians. The goal for the PACE system therefore, is to allow clinicians to create patient records that can be shared between their colleagues without the need for private patient details to be stored on the cloud. In order for users to share confidential information effectively, patient records must be divided with a subset of the data kept off cloud, on the users machine. P2P technologies can then be investigated as the method to most efficiently transfer these private details between users.

To facilitate the sharing records between users, the real world model of communication between clinics and their clinicians was adopted as the topology for sharing private data. Clinicians of the same clinic share their information automatically and compile a shared store of records. If a clinician requires records for a patient that is not a part of their clinic, they can send requests for this data to find a clinic that does have the patients records which can then be sent to the clinician. This same system of communications was modelled and mimicked in the design for the PACE system. Each user represents a clinician on the system that is associated with a clinic which has multiple other clinicians. These clinicians form and share their own distributed table of records. Any requests for other records are routed to other clinics on the system and transferred to the requesting clinician.

The design for the PACE system is a variation on the MVC (Model-View-Controller) design pattern known as the MVP (Model-View-Presenter) pattern [8]. The *view* of the PACE design is the front-end point of interaction for the user on the client application. This client application includes the Graphical User Interface which displays

patient records and is used to create new records. The *presenters* of the system exist on both client and cloud ends of the PACE system. The cloud *presenters* translate the user instructions and send function calls to the models of the clinics, clinicians and the anonymised patient information. The confidential information is modelled and controlled on the client-side. However, there also exists another cloud-based *presenter* which controls the forwarding and receipt of requests to these client-side *presenters* for private patient details and enables the P2P communication between peers. The P2P network is an overlay of the modelled clinic-clinician configuration. Each peer on the network as a corresponding clinician modelled on the cloud, and each clinic is represented by a super-peer which can relay requests and responses across the network. Thus, all the patients assessment details can be kept on the cloud, giving the system the traits of accessibility, portability and availability. To enable the same set of features for the private patient details, a P2P network is used to allow users to share details directly with one another from machine to machine. The design and architecture of the PACE system will be described further in later sections of this chapter.

3.2 Requirements

The list of requirements, as detailed below, include all the obligations of the system. The system must have the ability to manage basic interactions with patient records as well as the more complex issue of sharing private data between peers.

- **Store Patient Records on the Cloud.** The most basic requirement for the system is the ability to store patient records on the cloud. All users should be able to store records to the same central database on the cloud.
- **Authenticate Users.** Once records are on the cloud, before they can be retrieved, the system must ensure that users are identified and authenticated. This is to ensure that only registered healthcare workers have the ability to see and find patient records kept on the cloud.

- **Review Records.** Users must be able to then view patient records on the cloud that both they and others have entered onto the system.
- **Protect Patient's Identity.** Due to concerns relating to the safety of a patient's identity on the cloud, the system must have some means of negating the risk of private patient details being kept on the cloud.
- **Share Data Between Users.** Records on the cloud should be viewable to all users that have gained appropriate permissions. This includes all confidential patient information.
- **Search for Patients.** Users should be able to search the system for a record they do not have using some means of identifying the patient.
- **Query Patient Database.** The system should also extend beyond searching for singular patients and allow the database to be queried. This allows multiple records matching a set of criteria to be delivered to the user.

In order for the PACE system to become a viable solution to the problem tackled in this research, each requirement must be fulfilled. The requirements for adding and retrieving patient details from the cloud and authenticating the user are relatively basic requirements and so do not require much explaining. The most challenging problems presented by these requirements are sharing patient data between other users, searching for patients across the system and querying the database of patient records. As the database of patients are divided and distributed according to our hypothesis, implementing these features becomes incredibly challenging which our design must address.

3.3 PACE Design

This section looks in greater detail at the design of the PACE system including the use of the MVP pattern as the basis for the design, how the data in the system is

handled, and how the P2P communications are used to enable the sharing of patient records.

3.3.1 System Design

As mentioned in section 3.1, the PACE system was designed based on a variant of the MVC pattern, the MVP pattern. The MVC pattern is useful to architect interactive software systems where it is necessary to create a separation between the user's view of the system and the underlying data [33]. MVP is an evolution of this MVC pattern which better reflects the modern web application by completely separating the *view* and *model* and having the *presenter* translate interactions on the *view* to a *model*'s functions and updating the view with the changes to the *model*. The PACE System was designed using this MVP pattern as the basis for the cloud-based application to fulfil the basic functionality of storing and retrieving patient records. The system can thus be broken into three distinct categories: the *Models*, the *Views* and the *Presenters*.

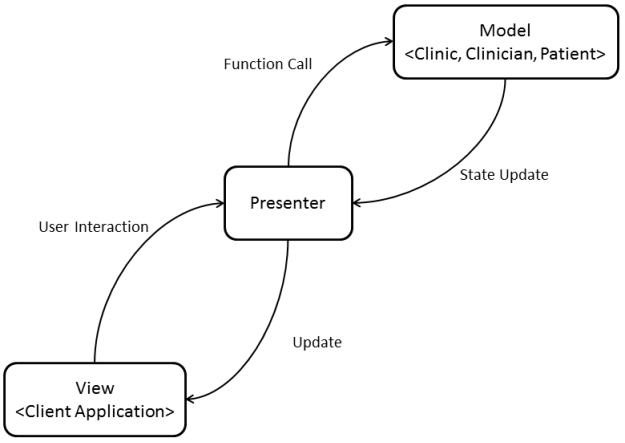

FIGURE 3.1: Model-View-Presenter Design

3.3.1.1 Models

The three most important classes in the PACE system are the `Clinic`, `Clinician` and `Patient`. As can be seen from figure 3.2, these classes are placed in a hierarchy

of control, similar to how a healthcare office would run in reality. In order to operate with these models, the system uses Data Access Object (DAO) instances which control all interactions with the underlying data. There are two DAOs used by the PACE system, both which implement a DAO interface with the basic CRUD (Create, Retrieve, Update and Delete) functionality. The `ClinicDAO` implements this interface and manages the administrative objects on the system. This DAO can also be used to retrieve, add and remove clinicians from clinics. The `PatientDAO` performs operations on the patient data and also allows for new records to be associated with patient and can run SQL-like queries on the patient data.

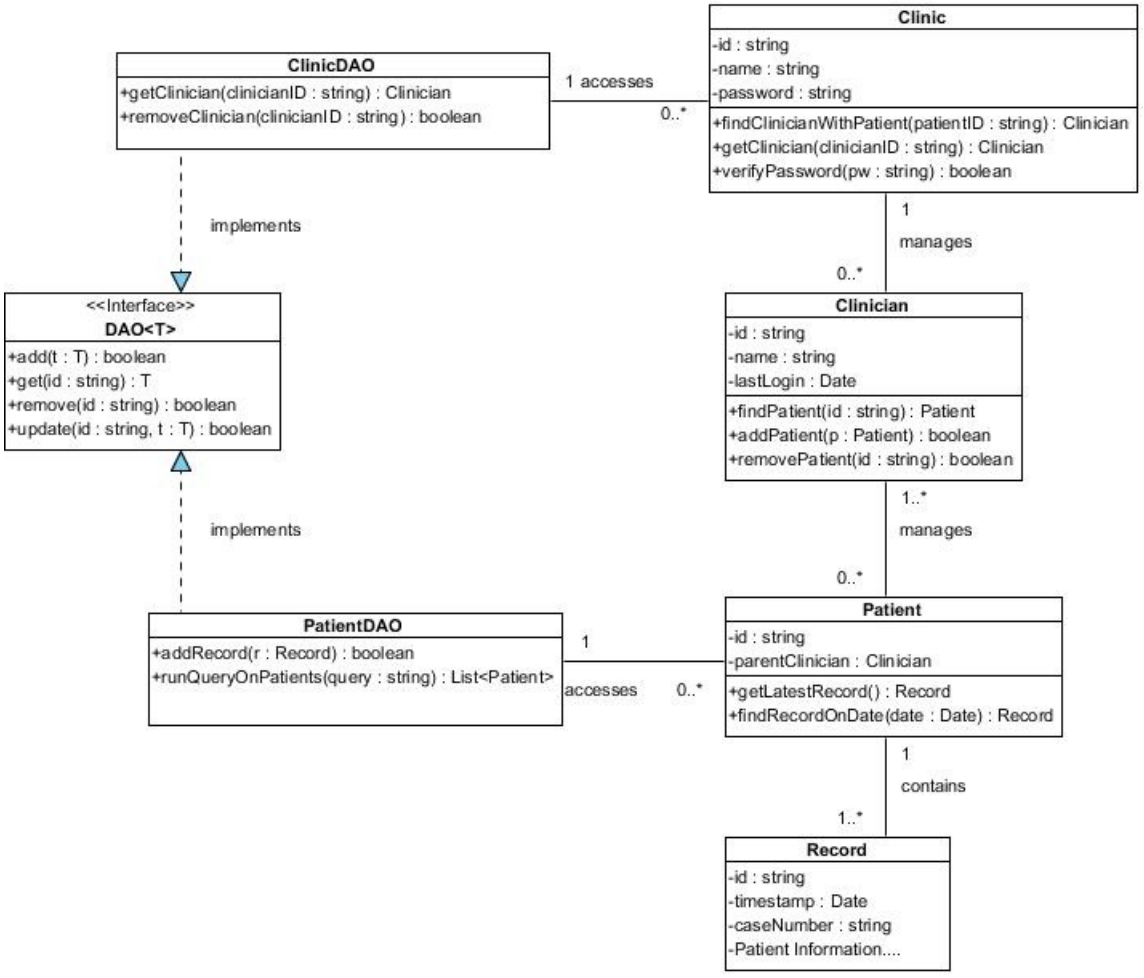

FIGURE 3.2: Class Diagram of Cloud Design

Clinic Each `Clinic` on the system must have an ID and a name in order to be found and identified on the system. A `Clinic` object also contains a hashed password

variable which can be used to verify a clinician belongs to a `Clinic`. A password can be checked using the `verifyPassword` function. Each `Clinic` also manages a number of `Clinician's` that belong to it. These `Clinician's` can be found and queried with functions `getClinician` and `findClinicianWithPatient` .

Clinician A `Clinician` represents all registered users of the system and must belong to a `Clinic`. Each `Clinician` must also have a unique id and a name. A `Date` object is also used to log the last activity of the `Clinician`. There are also functions that are used to manage the list of `Patient's` the clinician currently has associated with it, according to the PACE system.

Patient and Records Each `Patient` on the system has an identifier that can be used to anonymously identify the `Patient` throughout the PACE system. It also contains a reference to the `Clinician` that first entered the `Patient` onto the system. Each `Patient` also contains one or more `Record` objects. These `Record` objects contain the `Patients` medical information entered by the clinician. As a `Patients` state can change over time, there exists a need to maintain a history of the `Patients` records, so multiple records can be associated with each `Patient` object and differentiated through their date of entry. Each `Record` contains an identifier, a timestamp, and the medical information relevant at the time of entry.

3.3.1.2 View and Presenter

The *view* and *presenter* manage the user input and how it affects the state of the Models on the cloud. The *view* is the GUI of the application on the clients machine; the interface through which the user can view and alter the *models*. In typical MVC designs, once the *model* is updated via the *controller*, the *view* is changed directly to reflect the interaction. However, in modern web applications, the *view* has the potential to change what the user can see independent of other actors on the system. The MVP pattern supports this capability by entrusting the *presenter* with the responsibility to both translate user interactions to function calls on the *model* and

to also send any state changes to the *view*, so the *view* may decide what should be presented to the user.

The *view* in the case of the PACE allows the user to input patient details through various forms that constructs the `Patient` and `Record` objects on the system and the ability to see both the cloud and peer-stored patient details. The *presenters* in a web application usually take the form of the interface between back-end operations and the front-end interface, for example Java Servlets or a .NET HTTP handler. The main operation of the *presenter* in the design is to translate requests sent from the *view* into corresponding function calls on the back-end. Using DAO instances as the interfaces to the stored data, the *presenter* can perform tasks on both patient data and administrative data such as creating a new `Clinic` or finding a `Clinician` associated with a particular `Clinic`.

3.3.1.3 MVP for Basic Functionality

By utilising the MVP pattern, the cloud portion of the PACE system was designed to allow users to create and review patient records from a web application. User input affects the *view* which sends the corresponding instructions to the *presenter*. The *presenter* can then make the requisite changes to the stored data via the DAO's and then return the result of the functions to the *view*. When a clinician is attempting to add a patient, they can do so using a series of forms presented to them by the *view*. Once the forms are completed, the *view* then forwards the patient information to the *presenter* which translates the contents of the user input into the parameters necessary to create a `Patient` object. Using the `ClinicDAO` class, the *presenter* can instruct back-end services to add this patient to the database and update the `Clinician's` list of stored `Patient's`. Once done, the *presenter* informs the *view* of the successful addition of the patient.

3.3.2 Sharing via P2P

Although the MVP design pattern allows users to add and view their own patients on the system, the design must also incorporate a means to allow other clinicians to see the newly added data. In order to fulfil the given requirements, a means of sharing private patient information between users must be implemented as the client applications must also be able to review patients that have been entered by others. To do this, a network is overlaid across the system of users and allow P2P connections to be created for the purpose of sharing patient records without confidential data ever reaching the cloud.

To accomplish the sharing of patient data between users on over a P2P network, the model of clinics and clinicians is mirrored on a P2P overlay. Each clinician signed onto the system acts as a peer and can both send and receive data from other peers. To control the flow of data and to facilitate the ability to find patient data on the system, the peers are divided into groups dependent on their respective clinics. Each clinic is represented by a peer, which becomes the *super-peer*. The super-peer is an expansion of a peer with the responsibility of communicating with the network on behalf of other peers. This super-peer denotes a clinic and can forward requests to other super-peers in order to locate and retrieve the patient data from another clinic. Super-peers are used to help organise the peer network and allow for a measured approach to forwarding requests across the network. Instead of searching a web of peers for a single patient, the search criteria can be sent to each super-peer and dispersed to their peers.

3.3.2.1 P2P Model Constructs

With peers randomly connecting to the system and individually creating data, all peers (end-users) are organised into peer groups (the clinics). These peer groups are organised by super-peers and share private patient data between themselves on creation or update. As patient data generated by peers may be required by other

peers, it is important that the system has some level of data redundancy to ensure availability.

We begin with a set of assumptions:

- Identifying data is stored off-cloud.
- Each super-peer represents a clinic and maintains a group of peers that represent the clinic's healthcare workers. These peers and super-peers correspond to the models of the clinicians and clinics on the cloud respectively.
- Each peer is aware of the super-peer it belongs to but has no knowledge of the other peers that are connected to the super-peer. The only time a peer is aware of other peers is when the super-peer introduces one peer to another for the purpose of sharing patient data over a P2P connection. This communication is hidden from the user.
- All P2P connections are transient. Once the patient data has been shared, the connection is destroyed. This helps maintain the performance of the client application and reduces the risk of data being sent over dead channels.
- Users sign into PACE with a username and password and are then connected to the network via the super-peer.

Now we progress to describing the constructs of the PACE system in terms of their attributes and functionality, of which a subset is shown in figure 3.3.

- `Peer.` The attributes of the peer are its ID, Peer Context, and the super-peer group to which it belongs. The peer context is a representation of the peer's presence on the cloud, in this case it is the `Clinician` object, as mentioned in figure 3.2. The functions include: `makeRequest(patientId)` which sends a request to the super-peer for a particular patient's data; `Connect(PeerAddress)` creates a P2P connection with another peer; `sendPatient(Peer, Patient)` which sends requested patient data across the established P2P connection; and

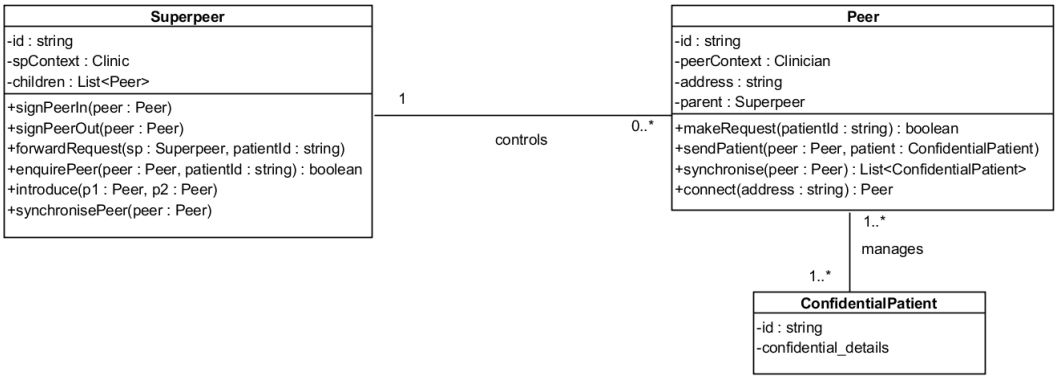

FIGURE 3.3: PACE P2P Model

the function `Synchronise(Peer)`, which compares the list of patients to another peer and updates its database.

- `Peer Context.` The Peer Context construct models the user of the system, a healthcare worker. This is the `Clinician` object from the cloud and includes the name of the user and the list of patients it has created.

- `Super-peer.` The attributes for the Super-peer are its Super-peer Context, a boolean variable identifying whether it is the controlling super-peer of the system, and a list of peers corresponding to the healthcare workers of a clinic. The functions of the super-peer are: `introduce(Peer, Peer)`, which begins the protocol needed to introduce two peers in order to set up a P2P connection; `synchronisePeer(Peer, Peer)` which uses the introduce function to connect peers it believes are not fully up to date with other peers; `enquirePeer(peer, patientId)` connects to a peer to check whether the peer holds a particular patient record; and `forwardRequest()` creates a connection to another super peer to find private patient data stored by that clinic. `SignPeerIn(Peer)` and `SignPeerOut(Peer)` add and remove peers to the network respectively.

- `Super-peer Context.` The Super Peer Context represents the healthcare clinic. The attributes relevant to the super-peer are the name of the clinic and the list of clinicians that are a part of it.

These constructs form the basis for the P2P network that enables clinicians to share patient data between themselves. In order to incorporate such a network into the PACE design, the system requires some means of controlling and enabling the introduction of peers from different clinics across the network. To achieve this, a component should be used that helps keep track of peer actions. This component can be contacted by the super-peer to help locate patient data on the network and so must also be informed of newly *elected* super-peers if the previous super-peer leaves the network.

This P2P network not only allows private patient data to be shared between peers, it also allows requests for this anonymised data to be forwarded across the network. This fulfils the requirement for the ability to search for patients on the system, as mentioned in section 3.2. A request can be created by a peer on the system, sent to the super-peer which can forward this request onto other super-peers. Each super-peer can then ask its peers to search for the requested patient. If the patient is found, the peers can be introduced and the data shared, thus fulfilling the search.

3.3.3 Patient Records Management

Now that the overall design of the cloud application and the P2P overlay have been described, this section will detail how the records of the patients are managed throughout the system, from creation to removal. This will include descriptions of how the data should be separated and then how the data will be recombined once a request is received.

3.3.3.1 Storing and Retrieving Patient Data

Figure 3.4 shows a Data Flow Diagram which illustrates the states and operations of the patient records.

Step 1. The first step in the flow of data is the creation of the patient record by the user. Once the user has created a patient record on their client application, the

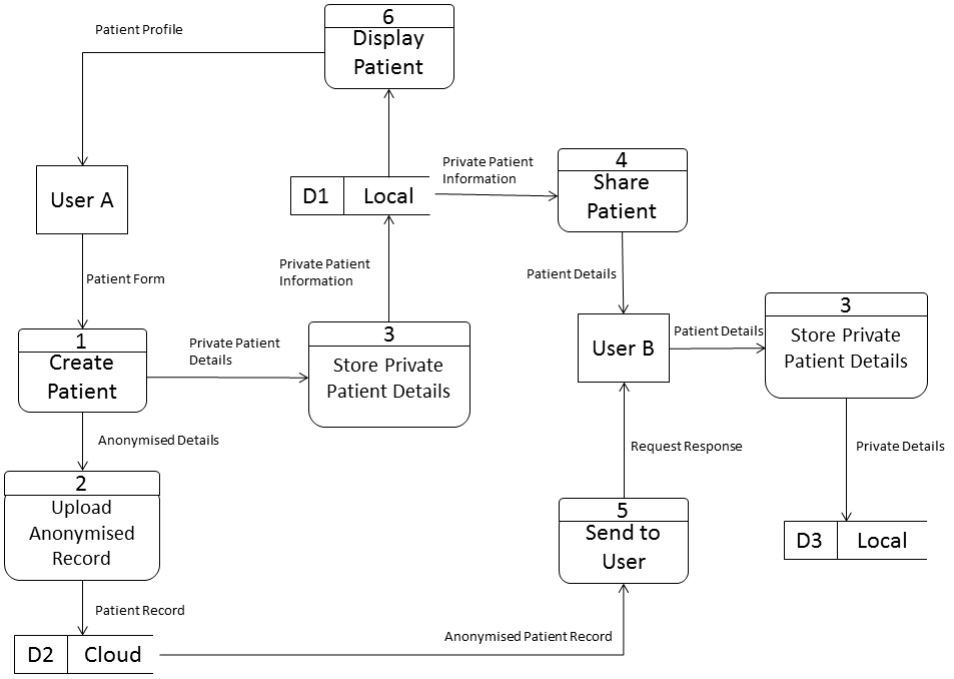

FIGURE 3.4: Data Flow Diagram

identifying attributes must be extracted. These are identified with the help of a clinician during the implementation stage of the design and include values such as name, PPSN (social security number) and any contact details. A unique key can then be generated using values from this set of identifying features. The key should be generated using an asymmetric cryptographic technique that would make it difficult to use the key to reverse engineer the identifying data.

Step 2. Once the identifying attributes have been extracted, the anonymised details are sent to the cloud with the generated key which will be used as the primary key on the system. As mentioned in the design of the cloud application, each patient on the cloud can have multiple records attributed to them.

Step 3. The extracted private data is then stored by the client on the user's machine. There should only ever be one set of values for a patient's personal information, if there are any changes to this data, the changes must be pushed to all other peers on the network that store that patient.

Step 4. When a user successfully stores the patient details on their database, they then share this information with other peers in their peer group over a P2P connection. The sharing process is also performed when a user (User B) requests the data from another user (User A). This can either be done by supplying the information necessary to regenerate the primary key for the patient (name and date of birth, for example) which is then used to locate a peer with the patient and initiate the sharing. Alternatively, other information can be used to form a search query which is distributed across the peer network. Any patient that matches the search query can be sent by a peer to the user.

Step 5 (Optional). If a user (User B) seeks to retrieve the patient records, they need only provide the primary key that was generated on creation. Using this key, the cloud can retrieve the patient object and send the records to the user. The two sets of data - confidential and anonymous - can then be joined on the shared primary key. However, if a different user makes a request for the data, the information must be retrieved from both the cloud and the user.

Step 6 (Optional). If a user needs to review the private details of a patient to be combined with the records received with **Step 5**, confidential data can be retrieved from the local database and displayed on the client application.

3.3.3.2 Propagation of Local Updates

As confidential data is distributed among multiple peers, when one peer updates the confidential data (changes the patients contact information for instance), propagating this change can be problematic. To enforce data concurrency across the peers, a basic version control mechanism should be introduced. When a user updates confidential data, the patient object is given a timestamp at the time of alteration. The peer then informs the super-peer that the patient has been changed and the super-peer instructs all of its peers to request the update from the originating peer. Whenever a peer (Peer A) logs in, they are introduced to the peer with the longest currently running session (Peer B) and their data is compared. As the data is being compared,

if the elder Peer B has a patient object with a more recent timestamp than Peer A, it sends that patient object to Peer A. The odds of an update conflict are minimal as the system is for managing patient data, so it is assumed that if a clinician is changing confidential patient data then the patient must be providing this new information and so should not be able to provide contradicting information to another clinician at the same time. If however, such an unlikely circumstance occurs, the PACE system adopts the practice of enforcing the most recent timestamp.

3.3.3.3 System-wide Data Removal

A user can remove a patient from their machine and the system. If a user deletes a patient from the system, the patient is first deleted from the cloud then the user's machine. When a patient is removed from the cloud, the values are nullified and a `Deletion` object is left in its place. This `Deletion` object contains two variables. The first variable is equal to the number of clinicians that are a member of the deleted patient's clinic. Each time one of these clinicians logs in, they are told to remove the patient and the variable is decremented. Once the counter is down to zero, it is assumed all necessary clinicians have deleted the patient and the patient object is completely removed from the database. The second variable is a contingency plan in case a clinician does not log in regularly. The second variable is an expiration date; once this date is reached the patient is completely removed regardless of the first variable. If a clinician attempts to request data using an expired primary key, they are informed the patient does not exist and instructed to delete any patient potentially stored by that user.

3.3.4 Design Summary

This section has described the design of the PACE system and how it should operate to accomplish the requirements set out in section 3.2. The cloud application achieves most of the functionality of the requirements, such as adding and reviewing patient information. It also enforces the authentication of users with the clinic passwords to

ensure the patient has the authority to be a part of the clinic, although a means of identifying the user on the system will also be needed. By separating the data and storing anonymised data on the cloud and identifying data on the users machine, the design protects the patients identity. The P2P network of users satisfies the requirements to share patient data and the search for patients on the system. The last requirement of the PACE system is the ability to query this divided database of anonymised and identifying records. The details of how the PACE system accomplishes this is described in Chapter 4. First, have the following two sections describe the architecture used to implement this design.

3.4 Cloud Architecture

In the previous section, the design for the PACE system was described. In this section, an architecture is introduced which aims to implement the cloud design. As mentioned previously, the PACE system uses a novel combination of cloud technologies and a decentralised hybrid P2P topology, as described in [4], to enable the sharing of confidential patient information between healthcare professionals working in hospitals and clinics. The cloud portion of the system manages user operations via HTTP requests from the client and the systems data is kept on cloud using an SQL database. Figure 3.5 demonstrates the design of the cloud portion of the PACE system, which has four main components (P1 - P4) which will now be described.

3.4.1 Client Interface (P1)

The Client Interface component is the primary interface between the PACE system and the user. There is a basic set of functionality at this level, and in most cases, the work is performed by similar functions in the back-end processes.

- `Authenticate`. This function authorises users attempting to sign in and also informs the super-peer that a peer has logged on and is now available to send

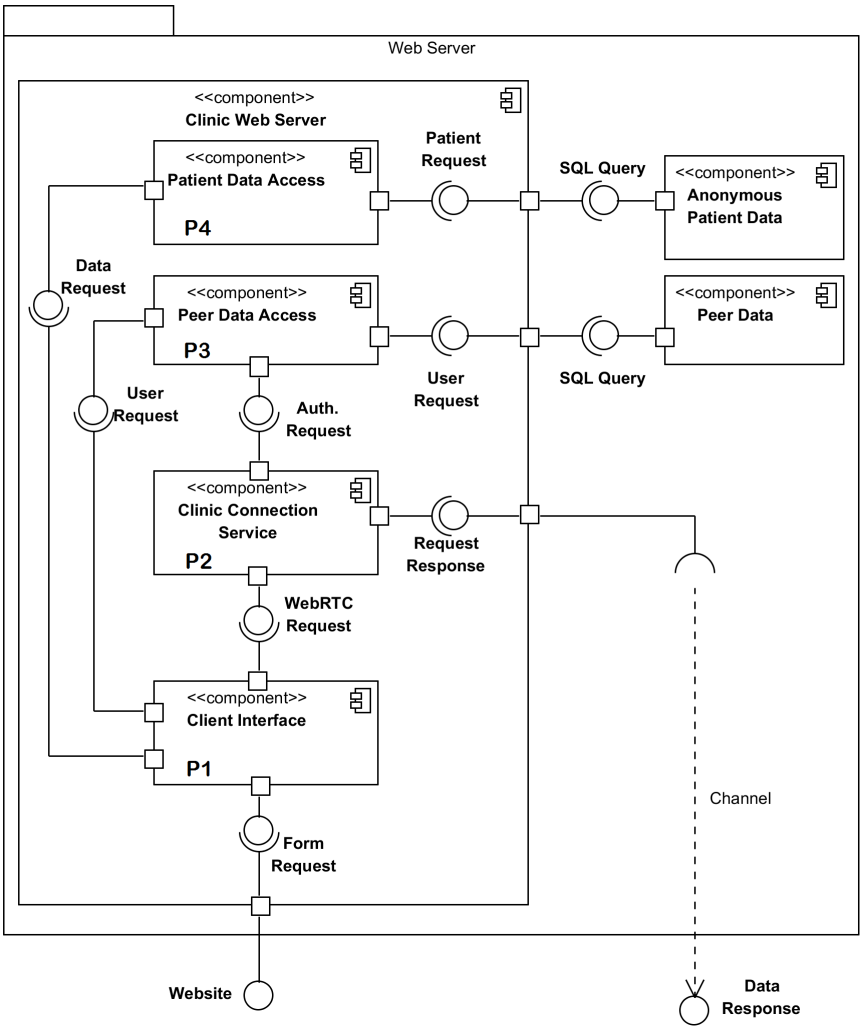

FIGURE 3.5: PACE Components and Interaction

data. An email account is used to authenticate each user before interaction with internal components takes place.

- `Search`. The function uses the Clinic Connector (P2) to find the location of a patient's private data and initiate a P2P connection with the hosting peer. The search function is the first step in the peer to peer sharing protocol described in Section 3.3.2. A patient identifier is required to search for a patient on the peer network. A clinic identifier can also be used to reduce the patient search to a single clinic.

- `Retrieve`. This function interacts with the Patient Data Access (P3) component to find and retrieve patient data stored on the cloud. Similar to the Search

function it requires a patient identifier and optionally, a clinic identifier.

- `Update`. The Update function interacts with the Patient Data Access (P4) to either update or add a patient's record on the cloud. As a patient is being added to the system, the client interface receives a unique identifier for this patient data from P4. This identifier is used when retrieving both the patient data stored on the cloud and by a peer group. If a new patient is added the Client Interface interacts with the Peer Data Access and Client Connector components to initiate the synchronisation of data between the peers.

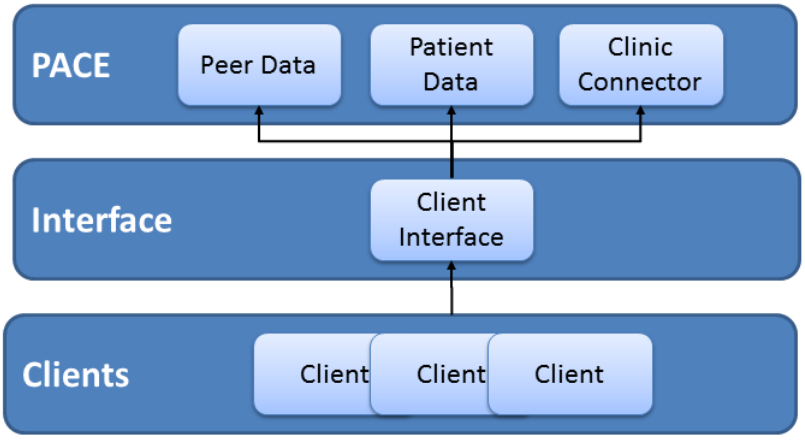

FIGURE 3.6: PACE Client Interface

3.4.2 Peer Data Access (P3)

This component controls all access to the administrative data stored concerning the clinics and clinicians on the system and their connections to the P2P network beyond the cloud. The Peer Data Access component fulfils the responsibilities mentioned in the design in section 3.3 for the `ClinicDAO`. As mentioned in that section, the `ClinicDAO` is responsible for managing the data and relationships between the clinics and their clinicians. The Peer Data Access component achieves this by acting as an interface between the Client Interface and the database containing the clinician and clinic information.

3.4.3 Patient Data Access (P4)

Just as the Peer Data Access component implements the responsibilities of the `ClinicDAO`, this component achieves the same for the `PatientDAO`. The goal for the Patient Data Access component is to manage the anonymised patient data stored on the cloud using the Java Object-Relation Mapping library Hibernate to interact with the database. All functions are called by related functions from the Client Interface component.

- `Patient_Search`. This function locates specific patients based on a unique identifier and optionally a clinic identifier to reduce the dataset to search.
- `Query_Search`. This function is much broader than the standard search function as it provides the ability to use queries on the cloud based data. Multiple records are returned based on the criteria specified in the provided query. Queries for the data are translated into HQL, a SQL-like query language for Hibernate databases.
- `Update`. This function updates a patient in the database, or if the patient does not currently exist, it inserts the new patient's records.

3.4.4 Clinic Connector (P2)

The Clinic Connector service locates and connects users across the network. In other words, it acts as the handshake mechanism to introduce two clients in order to initiate a transient P2P connection. This has two scenarios: when a peer has been found to be missing newly added patient information; and when a user makes a request for patient information from another clinic. If a peer is missing information from the clinic, the Clinic Connector introduces the peer to another peer connected to the same super-peer. The super-peer chooses the oldest peer on the network to update the new peer, as it is assumed to have received the most updates compared to the other peers. The peers can then swap stored patients and update each other's records

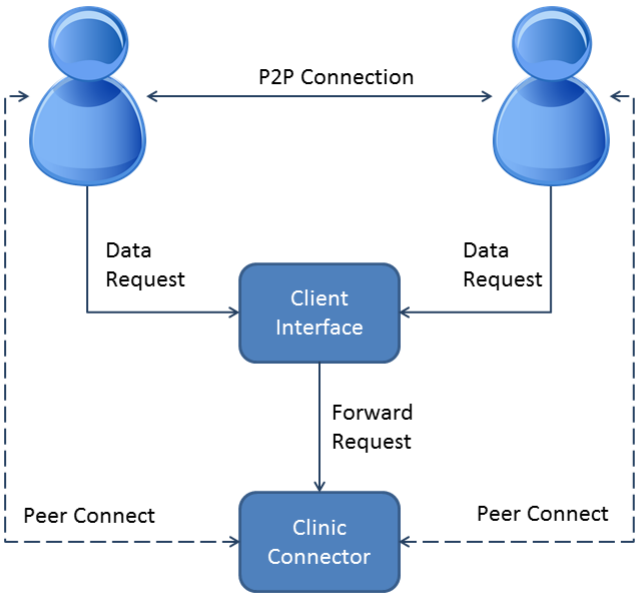

FIGURE 3.7: PACE Clinic Connector

of patients. If the clinician makes a request for a patient from another clinic, the Clinic Connector instead communicates with multiple super-peers to find a suitable peer that contains the information. To find a suitable peer, the Clinic Connector utilises the Peer Data Access (P3) service.

In order to introduce two peers, the Clinic Connector is required to communicate with each client. A direct communication is formed between the Clinic Connector and the client application that allows the cloud to send updates and requests without the need for routine polling. The method used to accomplish this is discussed in 3.6.

3.4.5 Server Architecture Summary

This section described the architecture for the PACE system which implements the design described in section 3.3. There were four main components to the architecture: the Client Interface, the Clinic Connector and the Peer and Patient Data Access components. These four components operate together to allow clients to send, retrieve and search for patient information via HTTP requests and can also update client

applications with update instructions. The cloud also serves the use of acting as a central server to allow peers to discover their network address and aid super-peers in introducing peers to form P2P connections. In the next section, the architecture for the client application portion of the PACE system is described.

3.5 Clinic and User Clients

In this section, we provide a description of the different client components and briefly explain how these components facilitate interaction with the PACE system. Recall that there are two forms of data: identifying data which is always stored locally and anonymised data which can be stored locally or on-cloud. The system must accommodate for two different states of the client in regards to this anonymised data. The first state is the default state, in which all the user's anonymised data is stored on the cloud and only the identifying fields are stored locally. The second state occurs when a user inputs data relating to a patient but must also cache the anonymised data locally, perhaps due to the user having no internet access to store the information. This cached data is kept by the user until they next log in to the system when it is automatically uploaded to the cloud and the user returns to the default state.

Figure 3.8 demonstrates the design of the PACE client which communicates with the PACE system on the cloud. The three main components (P5 - P7) to the client will now be described.

3.5.1 Client Application (P5)

The Client Application is the point of entry for the users and is the View portion of the MVP design pattern mentioned in section 3.3. The Client Application processes all the user input and forwards the necessary requests to the server and the other local components in order to create, remove, update and search patient files both locally

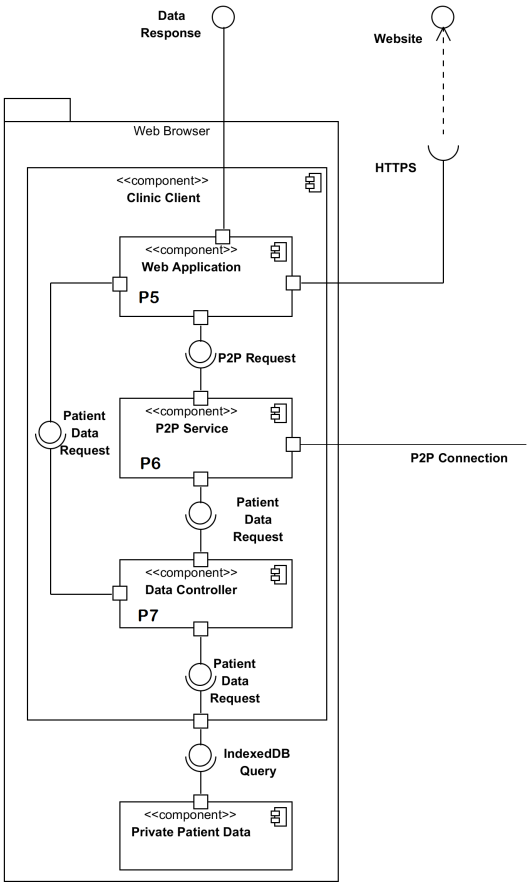

FIGURE 3.8: PACE Client Component Diagram

and on the cloud. In order to accomplish these tasks, the user's input is translated and formed into SQLite-like queries that are parsed and translated to functions on the server. SQLite is a restricted form of SQL. Details of how these queries are formed and parsed are given in chapter 4. The following are the functions of the Web Application:

- `Create_Patient`. Uses form data filled in by users to create the patient data stored by the system. Any data that could potentially identify a patient is stored locally through the Private Patient Data Access component (P7), and the rest of the data is sent to the PACE system on the cloud to be stored.

- `Find_Patient`. This function finds patient data stored locally by the client. Using the Private Patient Data Access component, it searches the local database using a patient identifier to find an individual patient.

- `Query_Cloud_Data`. This function sends a query request for a subset of cloud stored patient data. The identifiers of the patients are not needed as it simply returns a subset of the anonymised data corresponding to the criteria of the query.

- `Request_Patient`. This function sends a request to the PACE system for a specific patients using the patient identifier (and clinic if available). If the patient is located, the client receives a response containing the network address to a peer willing to send the data. The user need not be concerned with patient data location (locally, local to clinic, or remote) as this is resolved internally.

- `Retrieve_Patient`. This function retrieves the anonymised patient data stored on the cloud for a particular patient. This patient's identifier is sent in the request to the PACE system. The data retrieved should correspond to a patient stored by the client.

3.5.2 Peer Connector (P6)

The Peer Connector component of the client is responsible for establishing connections with other peers and sending/receiving patient data on a peer level. The PACE system clients create transient P2P connections in order to share private patient data. The Peer Connector has two main functions: `Send_Patient`, which sends private patient data over the P2P connection, and `Receive_Patient` which stores patient data sent from another peer. The methodology used to form these connections is described in section 3.6.

3.5.3 Private Patient Data Access (P7)

The Private Patient Data Access (P7) component has much of the same functionality as the Patient Data Access component on the cloud. However, here the component works with confidential patient data. This data is stored locally in key-value pairs.

The component can store, remove and retrieve patients. The data store also supports the use of queries to return multiple patients matching a set of criteria. The key used to store patients locally, matches the primary key on the database on the cloud to allow the two sets of data to be easily recombined. Section 3.6 details what mechanisms are available to implement this client-side storage.

3.5.4 Client Architecture Summary

This section described the components and their respective processes within the client application. The Web Application is the main point of entry for the user and provides most of the basic functionality of the client. The Peer Connector component connects with other peers on the network to send and receive confidential patient information. The Private Patient Data Access controls access to the private patient data stored on the user's machine. Both halves of the architecture have now been described and the next section describes the technologies that can be used to implement the architecture and which were preferred.

3.6 Understanding the Technologies

In this section, the methods used to enable the PACE system to tackle the main challenges presented in the design are all described. The biggest challenges in the implementation of the PACE system are: implementing a P2P network using only web browser technologies; storing large numbers of private patient records on a browser; and supporting such a system with a cloud application as the server. In order to implement such a cloud application, a great deal of effort was given to choosing the correct 3rd party cloud provider for the task. The process of choosing the provider is now described in section 3.6.1.

3.6.1 Cloud Provider

The decision on which cloud provider for the development of the prototype was made on the basis of 7 metrics: data storage, user authentication, security, server location, client communication, development environments and the cost or running the application. The choice was made between, three of the most popular cloud providers for PaaS (Platform as a Service) platforms: Amazon[59], Google[23] and Microsoft[58]. Below are the descriptions of each metric which contributed towards the decision making process and the respective comparisons between the three providers.

Data Storage. As there would be large amount of patient data that needed to be queried, it was important that the chosen cloud provider had a versatile and reliable data storage service that could accommodate large searches and SQL-like queries. Microsofts data storage options include various standard storage options such as temporary local storage for the virtual machine running the application and the more permanent cloud storage. It also includes access to Microsofts SQL database which can be accessed via Microsofts SQL server software. Amazons storage options include their RDS solution, allowing instances of Microsoft SQL servers to be deployed via Amazon Web Services. Amazon also provide their own S3 storage solution, for large amount of schema-less data. Similar to both Microsoft and Amazon, Google includes solutions for storing both schema and schema-less data with the services Google Cloud SQL and Google App Engine Datastore respectively. All three parties provide similar solutions for data storage, with no party excelling above the others.

User Authentication. The requirements for the application necessitated a means of identifying and authenticating users attempting to sign into the system, to ensure the patient data stored on the cloud is safe from unauthorised access. Microsoft allows for the creation of an Active Directory on the Azure application, allowing users to be added and different access control levels to be attributed to each user. However, new users must be entered via a web portal to the service, which is not suitable for systems requiring minimal interaction

with the backend operations of the system. Amazon offers a solution similar to Azures Active Directory called AWS Identity and Access Management, allowing users to be created and assigned access privileges manually. AWS also allows a federated sign in process, so users can sign in using an Amazon, Facebook or Google account to identify themselves without needing to be entered manually. Google doesnt offer an access control service such as the Active Directory, but one can be implemented using the Spring security library [56]. Identity can also be verified using a federated login service allowing users to login with Google, Yahoo, Microsoft, LinkedIn and others. Although important for later implementations of the application, the user authentication for this research does not carry much weight in the choice of provider, but under this metric Microsoft and Amazon are preferable.

Security. Beyond ensuring that the system users are authorised, there is also a need to ensure that the chosen 3rd party cloud provider is keeping all the data stored in the cloud safe from unlawful intrusions. All three major cloud providers giver assurances of security; promising 24 hour monitoring, DDoS protection, data encryption using the AES-256 protocol and data redundancy strategies. As all provided the same level and method of security, they are equally suitable for the implementation.

Server Locations. As there will be patient records stored on the cloud (anonymised or not) there is a need to know where these records are physically stored. Microsoft, Amazon and Google all provide an option to choose where data should be stored, including data centres in Ireland. However, Google has this European option available only to those paying for a customer support package subscription. For those not paying the $150 dollar per month fee, the data and application is stored on servers in the United States. Both Amazon and Microsoft however, allow the Irish data centre to be chosen without an extra fee. Google is enrolled in the Safe Harbour program, though, which ensures that any data transported from Europe to America is treated with the same

protection laws as if it was being kept in Europe. Both Amazon and Microsoft are preferable to Google for the location of the patient data servers.

Client Communication. One of the requirements for the design is the ability for the server of the application to make unsolicited contact with a client in order to make P2P-related data requests on behalf of other peers. Microsoft offers a means of accomplishing this using their Message Bus system which allows for loose-coupling communication between server and client on a message system. Such a requirement can also be accomplished through the the use of Googles Channel API which allows for the creation of a persistent connection enabling an application to update a client immediately. Amazon does not provide a method of accomplishing this though an API or service, but it does allow scope through the use of web sockets. Using the Channel API to accomplish client communication is easier to implement than using web sockets and also does not require permission settings be applied via a portal, as is the case with Microsofts Message Bus. For this reason, Google would be favoured over both Amazon and Microsoft when communicating with a client.

Development Environments. All three cloud providers allow for the use of multiple programming languages when developing a cloud application. Microsoft allows the use of Java, Python, Ruby, PHP and their own .NET framework to create applications for Azure however, the application must be built using Microsoft's own IDE, Visual Studio and the use of many of their APIs and libraries requires special permissions being given to the application via various portals. Google supports the use of Python, Java, PHP and their own programming language Go to develop for the App Engine. No IDE is strictly necessary for the App Engine but the Eclipse IDE is advised by Google. Amazon supports Java, PHP and Python as well, but also Ruby and .NET. Amazon also provides support for developing and deploying AWS applications using Visual Studio and Eclipse. As the programming language of choice for this project is Java, all providers are viable options. However, in order to use Azure effectively, the .NET framework is needed for large portions of the server implementation and

the IDE for the purpose of building an Azure application is more convoluted than Eclipse. With this in mind, Googles App Engine environment appeared the best option for quickly implementing the prototype.

Another concern for the developer environment, is whether there is a way to test the app before full deployment and a payment is required. Both Google and Amazon allow applications to be tested locally for deployment and also offer a free quota for certain services and resources so a small scale version of the application can be tested online. Microsoft however, requires payment details before applications can be run online or locally for development purposes. As Google and Amazon allow for an application to be tested for free both locally and online, they provide the environment of choice for prototyping.

Cost The price for utilising the services of these providers was one of the most important factors in making a decision. Each provider calculates the cost differently, so to demonstrate the difference, the price of running a set scenario between the three will be compared. The cost of running a server with 4GB of RAM and processing power of 1.6GHz and a database with 50GB of space is compared as a sample of overall price. According to the price calculator of Microsofts Azure [44], the price of running a Linux Virtual Machine on the Azure servers with 3.5GB RAM to run the application is €0.09 per hour or €66.49 per month. No option is provided to run a machine with 4GB of RAM with only one core, so two cores each with 1.6GHz will be running the application. To run a SQL database for a web application with 50GB is €93.74 per month. These two costs come to an annual charge of €1922.36 for using these services on the Azure platform. Amazon charges $50.67 per month for a Linux virtual machine on a single-core machine with 3.7GB RAM and $58.20 per month for 50GB of space on a SQL database [3]. The annual cost for running such a set up on Amazon is €966.24 annually. Google charges $16.98 per month for a single-core machine with 3.75GB RAM and $75.74 for a 50GB Cloud SQL database [24]. This comes to roughly €814.44 per annum. The cost of Microsofts Azure platform is more than twice the cost of running a similar application on either

Amazon or Google, making these the most cost effective solution for running such an application.

Provider	Storage	Auth.	Security	Location	Comm.	Env.	Cost	Total
Amazon	3	1	2	1	0	2	3	10
Google	3	0	2	0	2	2	3	12
Microsoft	3	1	2	1	0	0	0	6

TABLE 3.1: Weighted Metric Scores for Cloud Providers

In making the decision, each provider was compared using the above metrics. Each metric was then weighted based on the importance to the prototype being developed. For instance, both storage and cost were weighted 3 times as important as User Authentication, as for the prototype it was essential the cost of the application did not surpass a given budget and that the data on the cloud could be accessed easily and queried effectively by the user. Where the provider was the preferred choice, it was given a score equal to the weighted value of the metric. Table 3.1 gives an overview of the scores for each provider and demonstrates that for the purpose of this research, Google was the clear winner with Amazon an acceptable second choice. When comparing between the three, Google proved to be the most cost effective and best suited for client communication and so was chosen to implement the PACE system.

3.6.2 Establish P2P on a Client

In order for the clients on the system to share the private patient data with other users, a P2P connection must be constructed. As the PACE clients use web browsers, the means of accomplishing P2P communications between users is limited. WebRTC (Web Real Time Communications) is an online javascript library that can be leveraged for this very purpose [63]. It allows for the direct communication of video and audio data between browsers, as well as P2P file sharing without the need for a central server or plugins [60]. The API for WebRTC was defined by W3C (World

Wide Web Consortium) with the help of Google and has been implemented for the most part by both Google and Mozilla. As WebRTC is quite a new introduction into network programming, it has not yet had the support of all the major browsers but much of its functionality is supported by Google Chrome, Mozilla Firefox and the Opera web browser.

The PACE system uses WebRTC to transfer JSON packages containing the patient data between peers. Prior to data being transmitted from one peer to another, the peers need to know the network location of each other, in other words the IP address of the browser. This is achieved by sending a request to a dedicated ICE (Interactive Connectivity Establishment) server. ICE is a protocol to help applications, such as P2P ones, identify themselves on a network. Once the ICE server identifies a client and returns the IP address, a central server is required to share the addresses with the two peers. Once they have each others network address, they can send data to one another without the need for a central server.

3.6.3 Client Storage

Typically, data storage on browsers has been limited to the use of cookies. Recently, however, solutions aimed at providing storage for much larger data have arrived. Web SQL is a SQL-like database for web browsers and is supported by Google Chrome, Opera and Apples Safari. However, work on Web SQL was ceased by W3C due to a lack of independent work on the project [61]. HTML5 also provides a means to store data on the browser for either a single session or using the more permanent local storage [62]. This HTML5 functionality is supported by most up to date browsers, however, the single key-value method of storing data does not suit the needs of a storing a dataset of patient records as it would require a very large JSON object to be stored and retrieved for every read/write operation.

IndexedDB is an API for browser based storage of large sets of structured data and is capable of high performance searches on these datasets through the use of indexes [45]. IndexedDB is in the drafting stage of the W3Cs specification. Currently,

IndexedDB has support from most of the main browsers such as Chrome, Firefox, Internet Explorer and, from Fall 2014, Apples Safari. IndexedDB allows large sets of data to be grouped, searched, iterated and filtered which suits the needed functionality for this researchs patient records. A large set of records can be stored on a users browser and remain easily searched and packaged for transfer between users. IndexedDB also operates on a same-origin principle, preventing sites from different domains accessing the data. Information is stored in key-value pairs by IndexedDB, allowing us to use the primary key for both local and cloud stored data, no need for key tables for each patient and the same key can be used on every client, increasing interoperability. IndexedDB was chosen to implement the client-side patient storage as it allows for large data sets to not only be stored but also searched and queried, making it the most suitable option for storing large amount of patient records.

3.7 Conclusions

This chapter began with an overview of the PACE system and the problem it aims to solve, the sharing of confidential information that is not stored on the cloud between users. The PACE system keeps much of the data on the cloud while keeping the confidential data stored on the users own machine. This locally stored data is shared with others over a P2P network. The requirements for the PACE system were detailed and then followed by the design that aims to fulfil those requirements. The design section included the use of the MVP pattern for the cloud application and the details on the use of a P2P network to enable the sharing between users. The architecture of the PACE systems server was then detailed, with descriptions for the role and functions of each component. This was followed with a similar break down of the client application. The chapter also described the decision behind using Google as the cloud provider for the system and the use of WebRTC and IndexedDB to accomplish the P2P connections and local browser storage respectively.

One of the requirements mentioned in this chapter is the ability to query the database of patients. However, this is quite challenging as the PACE system divides the data

into confidential and anonymised values and stores them separate from each other. The anonymised data is then distributed amongst the peers, making querying the collective database difficult. To tackle this, SQLite-like queries are generated to fetch and combine data from the two divided sources. These queries are described in much greater detail in the following chapter.

Chapter 4

PACE Queries

In the previous chapter, the design and architecture of the PACE system was described in detail. As mentioned in the design of the system, the patient data is split into two databases. The anonymised Patient data is stored on the cloud while the confidential data is distributed across the peer network. This chapter introduces the method used by the PACE system to allow the data from both sources to be queried and combined. By forming SQL-like queries from user input and sending them over the network to the various databases, we can search for and join results from several sources into a single set of values matching the desired information for the user.

The structure of this chapter is as follows: section 4.1 introduces the queries used by the PACE system and the requirements of the generated queries. In section 4.2, the classification the queries are introduced, using their scope and operation as the determining factors. The lifecycle of the queries is described in section 4.3, which includes how the queries are formed, parsed and transformed. Section 4.4 then draws the chapter to a conclusion.

4.1 Overview

As has been discussed previously, the PACE system achieves patient anonymity on the cloud by keeping all identifying values on the local machines of users. Although the design presented in Chapter 3 provides a means of sharing this private information between users through web-based P2P communications, the problem of searching the entire database of patient information is challenging. Not only does the system require some means of querying multiple databases and combining the retrieved values, one of these databases is also distributed across a network of users that can be constantly changing. To address this problem, the PACE system uses a series of SQL-like queries to pass instructions and requests over the cloud and P2P network.

The PACE system manages a database of patient profiles, and in order to perform operations on these profiles, queries are used. A query can add or delete a patient from the database or it can also build upon a patient profile, remove parts from that profile or alter the profile. A patient profile is comprised of private identifying attributes (confidential) and a series of medical records (anonymous). A query affects the patient profile by moving through the system to the relevant databases and executing a series of tasks though function calls. Queries are executed on the cloud for all the anonymised information of a patient's profile while the peer and super-peer execute the same query in order to retrieve confidential patient information. The two sets of results are then combined on the client machine and presented to the user. This gives the impression that all the patient data is stored as one, and not dispersed across multiple sources. Before the method behind this is clarified, some assumptions for the system and queries need to be made and then the requirements of these queries and how they are to be treated can be laid down.

4.1.1 Assumptions

There are certain assumptions that must be made clear before the query mechanism is fully described. These assumptions are listed below:

- Each entry to the patient database has an unique primary key known as the Patient Identifier (PID). This primary key is the same for cloud and locally stored patients.
- The PID is generated by the client using the patient's personal information, cannot be changed and is unique for every patient. The identifier is generated using a trapdoor cryptographic mechanism that ensures the patient's identity cannot be discovered.
- For the purpose of our experiments and evaluation, it will be assumed the identifier can be generated using the patient's name and date of birth, as this information should never change.
- Every user on the system has the privileges necessary to access confidential patient data kept on other clinics. In practice, this may not be the case, but for the purpose of defining these queries, it is useful to assume so.
- Each patient can have multiple records attributed to them on the system.
- Unless stated, any alterations or requests for a patient's record will default to the most recent record saved.
- If the patient is stored locally by the user, it is assumed they have access to the PID without the need to query the data specifically for it.

4.1.2 Requirements

In order for SQL-like queries to operate successfully with the PACE system, there are a set of requirements that the query system must fulfil. These requirements are:

- The PACE queries must support all CRUD (Create, Retrieve, Update and Delete) operations.
- Both anonymised and confidential data should be accessible via the queries.

- The accessible confidential data should not be limited to that stored on the user's machine. The entire database distributed amongst the network should be accessible.
- In presenting the results of a PACE query, if both data sources were accessed, then they should be combined into one set of results.
- The PACE system should support the ability to return multiple patients that match the given criteria of results from queries.
- Any attribute of a patient, confidential or anonymised should be usable when specifying search criteria.

4.2 Classification of Queries

The PACE system classifies each generated query. This classification process is to quickly define the purpose of the query and the data that it aims to affect. Each query is classified using three categories.

Level of Confidentiality. As the queries used by the PACE system are targeted at distributed patient data, the first category of classification refers to the level of confidentiality of the data. Confidential data is any information that can be used to identify a patient and the anonymised patient data is the rest of the patient's records that cannot be used to ascertain an identity.

Query Scope. The second category of classification for a query is the *scope* of the query. The scope of the queries refers to the what areas of the system are affected and is defined with three levels: the peer level (individual clinician), the super-peer level (clinic or hospital), and the cloud level (overall system). For any query that targets the anonymous data, this query is classified as being part of the cloud scope. If the query requires confidential information using a patient identifier that already exists on the local machine, the query is at the

scope of the peer. If the user does not have that patient's information stored locally, the scope must be expanded to super-peer level. It is known as the super-peer level as it is now the responsibility of the super-peer to forward the query and return the results to the peer. The scope of the query is determined solely on the immediate effect on the data; any changes to the data made after the query has been run are considered to be as a result of the state change caused by the query and not the query itself. Therefore, if a peer is instructed to delete a patient's details because another peer removed it from the system, the scope is not expanded to the super-level as it is done after the query's lifespan. A query can potentially occupy multiple levels. If a single query requires both confidential and anonymised data it can occupy the cloud level as well as the peer or super-peer level.

Query Type. The final classification for these queries is the operation. A query can be one of four operations which are the standard for database interaction: create, retrieve, update and delete. A patient can be added to the system once enough information is provided to generate a patient identifier. A `SELECT` statement can be used to search and retrieve entries on the patient database. The database can be updated by changing a patient's confidential or anonymised values and a patient can be removed completely from the system. Only operations on anonymised data are permitted within the scope of the cloud, while on the peer scope both anonymised and confidential operations are permitted. This is to reflect the caching ability of the system. If a user is not connected to the internet, certain operations on the cloud database can be cached and executed once the user is reconnected to the system.

Table 4.1 demonstrates what level of confidentiality is applicable for what operation and at what scope. For instance, when updating a patient on the cloud, only anonymised data is affected. The following section describes each entry in the table in much greater detail.

Operation / **Scope**	**Create**	**Retrieve**	**Update**	**Delete**
Peer	Confidential & Anonymised	Confidential	Confidential & Anonymised	Confidential
Super-peer	X	Confidential	X	X
Cloud	Anonymised	Anonymised	Anonymised	Anonymised & Confidential

TABLE 4.1: Query Classifications

4.2.1 Query Classification Description

Create - Peer. This operation is used to add patients to the PACE system and are generated by the peer. This operation generates both confidential and anonymised data. The confidential data is stored locally by the user and the anonymised data is stored on the cloud. The create operation on the peer-level produces both confidential and anonymised data. The anonymised data can be generated and cached if the user is not in a state to send the instruction to the cloud while the confidential data is immediately stored locally.

Create - Super-peer. No entry is necessary for the create operation at the super-peer level as patients are not added to the network with a query. Any additions to the system using queries exist only on the peer and cloud. Once they are added, then the super-peer of the originator instructs the other peers to request updates for the new patient, however this exists beyond the span of a query.

Create - Cloud. The create query for the cloud scope is quite simple, it inserts anonymised patient data generated by a peer onto the system. This can be a new patient or to add a new medical record to a patient's profile.

Retrieve - Peer. The retrieval operation at the peer level is required to retrieve only data stored locally by the user. This local store includes the patient data stored on the local database and the data retrieved over the course of the session. To differentiate between the scopes of peer and super-peer, the query needs to

specify whether the request should be kept local to the peer or be extended. For an example of a peer level retrieval: if a user requires the contact info and names of all their clinic's patients, then as no information is required from the cloud or from other peers, the query only requires the confidential data from the local database.

Retrieve - Super-peer. The super-peer level retrieval operation extends beyond the peer scope by searching for patients across the entire network with the criteria specified in the query. Once the query specifies that the confidential search should check records that are not present on the clinics store, the query can be forwarded from super-peer to super-peer to find all patients that exist on the system.

Retrieve - Cloud. This cloud level query accesses the anonymised data stored on the cloud. No identifying values are requested or provided. A patient identifier can be included in the query to retrieve a single patient's records but is not necessary. The entire database of patients, regardless of the clinic they belong to, can be queried and results returned.

Update - Peer. The Update operation at this level involves changes being made to the confidential patient data stored by the peer. This change must also be transmitted to the other peers that hold the patient, which is done by informing the super-peer that the calling peer has a new version of the data. The details of how this is done have already been described in Chapter 3.

Update - Super-peer. There is no entry for the Update operation at this scope as there are no update queries that directly act upon the network of peers. Much like the Create operation, the Update operation on the peer triggers an instruction from the super-peer to other peers to request the update.

Update - Cloud. This entry is similar to the Update operation of the peer, but alters the values of the patient's anonymised medical records rather than personal information. For instance if a user is amending a mistake on a record, the Update operation allows for a value to be corrected.

Delete - Peer. This operation removes a patient from the peer's local records. The deletion operation at this scope only affects the database of the peer and does not extend to other peers or the cloud.

Delete - Super-peer. No delete query exists at the super-peer level as all deletions are made on a peer-by-peer basis as they are instructed. If allowed, it would be equivalent to deleting all records from a clinic or hospital department.

Delete - Cloud. The Delete operation at the cloud level removes all records of the patient on the system. This is a permanent removal of the patient's records. Once deleted, any requests for the data will return an instruction to the requestor to remove any record of the patient.

Once a query has been classified with an operation and scope, the system can progress onto the forming and parsing processes of the query. These are explained in the next section.

4.3 Query Lifecycle

In this section, the lifecycle of the query will be discussed. The stages of the query begin with the creation of the query, and depending on the type of query, it goes through several stages of parsing, the query is then transformed into a series of function calls and from this the result of the query is compiled and returned to the user. Figure 4.1 illustrates the steps involved in executing a query on the PACE system. Each stage of this diagram will be explained over the course of this section.

4.3.1 Query Construction

Queries begin their lifecycle being generated on the client application based on the input from the user. Each of these queries is based on SQL but reduced to a limited form of expressions in the context of the PACE system. For instance, the `FROM`

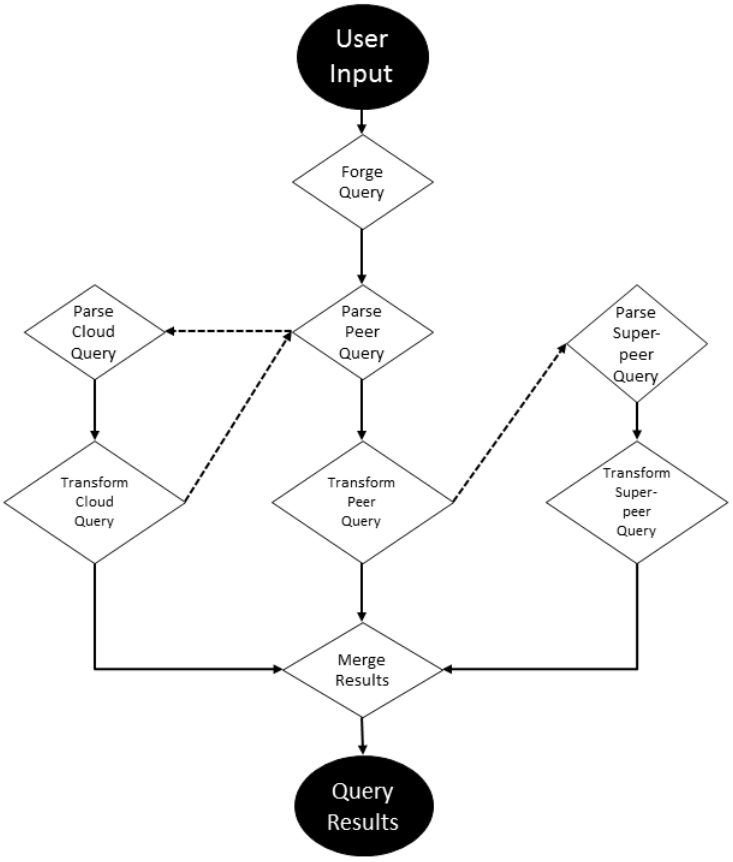

FIGURE 4.1: Query Lifecycle

clause is not needed to declare a table that the query targets, as only the patient database will be affected. Instead, this clause is used only when a particular record is needed by specifying the date of assessment or when the scope of the query must be explicitly declared by specifying a local search through the peer's database or a network search which involves searching the entire peer network. Both of these use case are demonstrated in Example 4.2. As mentioned in the previous section, there are four main query classes and each can operate at a different scope depending on the needs of the query. Each variant is detailed below including a template for the query and an example of a generated request.

4.3.1.1 Create Query

The creation query for the PACE system is a simplified SQL `INSERT` query, as demonstrated in Def 4.1. The create query can be used to create a new patient on the system

or add a new record to the patient's table.

```
INSERT
VALUES <Patient Values>
[WHERE PID=<Patient Identifier>]
```

Def 4.1: Insert Query Template

If a patient identifier is not stated using the `WHERE` clause, then a new patient will be inserted into the patient database. When a patient identifier is included, a new patient is not created, instead a new record for the patient specified by the identifier is inserted. All the values needed to create this query are acquired from information input by the user through the web interface. When a new record is added, the patient identifier is found by the system without input by the user. Example 4.1 illustrates an example of creating a new patient for the system.

```
INSERT
VALUES "Patient Name", "1/1/11", ...
```

Example 4.1: Insert Sample Query

4.3.1.2 Retrieve Query

The retrieval query is constructed using a `SELECT` query but with a few alterations to reflect the complexity of querying a central database joined with a distributed database. The template for retrieval is shown in Def 4.2.

```
SELECT [ID] <Result Columns>
[FROM <Local || Network> AND <Record Date>]
WHERE <Selection Criteria>
```

Def 4.2: Select Query Template

As can be seen in Def 4.2, the first line takes the shape of standard `SELECT` clause with the names of required columns. The only addition to this is the possibility of the use of the keyword `ID` preceding the columns. This `ID` keyword is added automatically when an identifying attribute is included in the `SELECT` or `WHERE` clauses. The keyword is used to inform the system that confidential data must be retrieved or used to filter

the results. If the `ID` keyword is not present, it is assumed the query is relevant only for patient data on the cloud.

The `FROM` clause is optional but can control two quite important details for the data retrieval. The first use is to specify whether the desired confidential data is sourced exclusively from the local database on the users machine (`Local`) or if the request is forwarded across the peer network (`Network`). The default option is to use the network for a larger dataset to query. A specific patient record can also be selected with this optional line. As each patient can have multiple records on the system, if a user wishes to obtain information regarding a specific record, the date of the record can be stated here and only information from the record with this date will be searched. If the `FROM` line is left out from the query, the default record is the most recent record added.

The `WHERE` clause is used to filter the result based on a set of criteria. The criteria set out can be applied to both the anonymised and confidential data. Example 4.2 demonstrates a sample retrieve query. It is used to retrieve the name, address and medical details of a patient with the name John Doe and date of birth January 1st, 1970. As can be seen in the query, the `FROM` clause is used to define the patient record that must be selected by specifying the date of assessment as the 15th of March, 2012. The example also demonstrates the use of the `ID` keyword to state that personal information is needed.

```
SELECT ID Name, Address, MMSE, Sleep_hours, Medication
FROM '15/3/12'
WHERE Name = 'John Doe', DateOfBirth = '1/1/70'
```

Example 4.2: Select Sample Query

4.3.1.3 Update Query

The update query is used to fix small errors in the data or make additions to the data for a record. It is near identical to a normal SQL `UPDATE` query, the only differences being the table name is omitted from the statement and the presence of

the `ID` keyword, to again signify that confidential data is included in the query. Def 4.3 demonstrates the template used to construct an `UPDATE` query.

```
UPDATE [ID] SET <Value Changes>
WHERE <Selection criteria>
```

Def 4.3: Update Query Template

Example 4.3 illustrates the use of the `UPDATE` query. For the patient with the specific identifier defined, the hours of sleep the patient gets is changed to one hour and the contact number is changed to '0755523'. The patient identifier is included automatically by the system based on the user input.

```
UPDATE SET sleep_hours = 1, contactNumber = '07555523'
WHERE PID = 'a572e47vf91ac'
```

Example 4.3: Update Sample Query for Changing Values

4.3.1.4 Delete Query

The `DELETE` query is defined by Def 4.4. There is only one difference between the query generated by the PACE system and the SQL `DELETE` query and that is the omission of the `FROM` clause. All the PACE system requires for the `DELETE` query is the instruction to delete and the `WHERE` clause that filters the table for the entries to be deleted.

```
DELETE
WHERE <Selection Criteria>
```

Def 4.4: Delete Query Template

Query 4.4 shows a simple example of a query used to delete a patient of the system with a patient identifier.

```
DELETE
WHERE PID = 'a572e47vf91ac'
```

Example 4.4: Delete Sample Query

4.3.2 Parsing Processes

Once a query has been constructed, the PACE system proceeds to parsing the query before translating it into function calls. The process begins on the client's machine, where the query was first formed but continues onto the cloud and the super-peer depending on the query. There are three steps to parsing a query: classification, attribute isolation and propagation. These steps are described below.

1. **Classify the Query.** The first step when parsing a PACE query is the classification, which has been defined previously in section 4.2, as the operation, confidentiality level and scope of the query. Every query is classified in the following order:

    1.1. **Operation.** The first category of classification that must be identified is the operation of the query. As described earlier, there are four possible operations which are the four most basic functions in persistent storage: create, retrieve, update and delete. The operation of the query decides the action that must be taken on the data selected.

    1.2. **Confidentiality Level.** Once the scope of the query has been identified, the values specified in the query can be investigated to determine the confidentiality of the query. There are two forms of data that the query is concerned with: confidential patient data and anonymised patient data. In order for the query to be parsed, it is important to isolate the values that are related to confidential data. When parsing a `SELECT` or `UPDATE` query, the presence of the `ID` keyword is a clear indicator of the presence of confidential data. If the scope of the query is at the peer and cloud level, the values must be divided between the two variants.

    1.3. **Scope.** Once the operation and confidentiality level are identified, the parsing proceeds to defining the scope of the query. The scope can be at the peer, super-peer or cloud level and can occupy multiple scopes at once. The scope of the query decides where the query must be sent in order to

retrieve the intended results. If the query is at the peer scope, the query can be run solely on the clients machine. However, if the cloud is also required, the query must be partially parsed on the client to form a new sub-query which is sent to the cloud. A similar sub-query must also be formed if the scope extends to the super-peer as well.

2. **Attribute Isolation.** Once a query has been classified, the next step is isolating and extracting any confidential information. The private patient details are extracted and used to generate a second query with the same operation but exist only on the peer and super-peer scope. The rest of the query should then be suitable for the cloud scope. In the case of a create operation, the confidential values of the patient are used to create the patient identifier that is sent to the cloud as part of the query. This identifier is used as the primary key for the patient.

3. **Query Propagation.** The next step in the parsing process is sending the queries where they need to be. As shown in figure 4.1, there is an order to distributing the queries. Parsing a query on the cloud takes precedence as it contains a full database that can be used to filter results much more effectively than the peer or super-peer.

    3.1. **Cloud.** Any query related to anonymous data is immediately sent to the cloud as the first step in parsing queries. On arrival, the cloud can classify the query itself to identify the necessary operation. If the operation is a `DELETE`, `UPDATE` or `SELECT` query, a filter is applied to the database of patients in order to match the criteria in the `WHERE` clause. If the query has an `INSERT` instruction, a new patient or record is created on the database. Once the tokens have been analysed, the query is transformed into a series of function calls. The transformation of queries into functions for the cloud, peer and super-peer is explained in section 4.3.3. The results of the query are then collected and sent back to the peer. If the query required changes only to the anonymous data, the results of the cloud can be presented directly to the user once they are received.

3.2. **Peer.** On occasions where the query does not require changes to the anonymous data, a query can be parsed and transformed by the peer immediately, otherwise, the peer waits for the results from the cloud. As the query has already been classified, the peer can begin transforming the query into function calls immediately. In the case of a `SELECT` query with a `FROM` clause instructing the query to be extended across the network, then the query is propagated across the network via the super-peer.

3.3. **Super-peer.** When a super-peer receives a query that must be parsed, the query is sent out to all other super-peers where it can be processed by other peers across the network. Any peers that find results for the `SELECT` query are introduced to the original peer and the results sent over a P2P connection. These results are combined with the current result set and duplicates are removed. The super-peer is the last possible step in the parsing process before the results are presented to the user.

As the cloud, peers and super-peers each parse a query, it is transformed into a series of function calls that achieve the intended effect of the query. It is these functions that create the table of results that will be merged and presented to the user.

4.3.3 Query Transformation

Transforming a query involves using the query's classification and keywords to call a set of functions in a particular order to fulfil the query's purpose. The following is a suite of functions for each entity that when combined, form the engine of the PACE systems query system.

4.3.3.1 Cloud

These are the suite of functions used in the execution of queries on the anonymous database of patients on the cloud:

- `createPatient(id:String, clinician:Clinician)` : Patient
- `createRecord(date:Date, <Medical Values>)` : Record
- `storePatient(patient:Patient)` : boolean
- `addRecord(patient:Patient, record:Record)` : boolean
- `getPatient(id:String)` : Patient
- `updatePatient(id:String, patient:Patient)` : boolean
- `removePatient(id:String)` : boolean
- `runQueryOnPatients(criteria:String)` : `List` $\langle Patient \rangle$
- `isolateSelectedValues(patients:List`$\langle Patient \rangle$`, select:String)` : `List` $\langle String[] \rangle$

The first seven functions in this list are all basic functions providing the CRUD functionality. The final two provide the means for querying the database for the purpose of searching for specific patients and retrieving selected subsets of data. The function `runQueryOnPatients(criteria:String)` applies a filter to the database of patients and returns a list of patient objects that match the criteria. The criteria is represented by a `String` which is formed from the query sent by a peer.

The last function on the list `isolateSelectedValues(patients:List`$\langle Patient \rangle$`, select:String)` is used to obtain a set of values from a single column of a set of patients. This can be run several times so multiple lists of column values can be joined into one result table that can be returned to the user. This function is used to retrieve columns of information specified in `SELECT` queries.

4.3.3.2 Peer

The functions used by the Peer are necessary for transforming queries at both the peer and super-peer level. The functions used at the peer level are concerned mainly

with interacting with the database of confidential patient information. The functions at the scope of the super-peer are primarily focused on sharing data between peers.

Peer Scope

- `addPatient(name:String, dateOfBirth:String,...)` : boolean
- `getPatient(id:String)` : Patient
- `getAllPatients()` : `List` $\langle Patient \rangle$
- `updatePatient(id:String, patient:Patient)` : boolean
- `removePatient(id:String)` : boolean
- `searchThroughPatients(patients:List`$\langle Patient \rangle$`, string: criteria)` : `List` $\langle Patient \rangle$

Similar to the functions on the cloud level concerning the anonymised patient information, most of the functions at this level perform the basic CRUD operations on the confidential patient database stored by the peer. The peer equivalent of the cloud function function `runQueryOnPatients(criteria:String)` to perform queries on the data is `searchThroughPatients(patients:List`$\langle Patient \rangle$`, string:criteria)`, except it explicitly requires the list of patients from the database in order to operate. Unlike the cloud variant however, there is no need for a function to reduce the number of selected columns. Due to the small number of columns on the confidential database, this functionality can be done at the time of presenting the results.

Super-peer Scope

- `makeRequest(sp:Super-peer, query:String)` : void
- `sendPatients(peer:Peer, List` $\langle Patient \rangle$`)` : void
- `compareWithPeer(List` $\langle Patient \rangle$`)` : void

The `makeRequest(sp:Super-peer,query:String)` function forwards a query onto the peer's super-peer to be dispatched across the network. No result is returned as the request is asynchronous. The `sendPatients(peer:Peer,List`$\langle Patient \rangle$`)` function sends a list of patients over a P2P connection to a peer. This is usually done once a request is received and the peers have been fully introduced. The last function is used to compare another peer's list of patients to their own in order to send any patients they are missing or not up to date.

4.3.3.3 Super-peer

All the functions used by the super-peer are concerned with discovering the location of data and connecting peers.

- `forwardRequest(query:String)` : void
- `enquirePeer(peer:Peer, query:String)` : boolean
- `synchronisePeer(peerA:Peer, peerB:Peer)` : void

The `forwardRequest(query:String)` function is used to pass on requests for patient data to other super-peers. This request is made to the clinic connector component on the cloud which communicates with the other super-peers on the behalf of the calling super-peer. When a super-peer receives a request for data, it forwards the query to its peers using the function `enquirePeer(peer:Peer, query:String)`, and the peers can respond with a boolean to indicate if they have the required information or not. The `synchronisePeer(peerA:Peer, peerB:Peer)` is used to update peers with the latest updates or additions made by a peer to their shared confidential database.

Once the query is transformed, the results of the series of functions are sent to the originating machine. All the results are combined then into a table of values. These results are then presented to the user via the user interface.

4.4 Conclusions

Over the course of this chapter, the queries used by the PACE system were introduced and an explanation was given as to how they were used to perform operations on a distributed cloud system. The classification of queries was introduced as a means for the system to quickly identify how the operation should be managed and where it should be sent on the system. The lifecycle of the query was then defined as the formation, parsing and transformation of a query over the various entities of the system which included the cloud, peer and super-peer.

The PACE system was created as an answer to the research questions posed in chapter 1. All the methods the system employs to solve these problems have now been introduced and detailed, so all that is left is to investigate the success of the PACE system. We now proceed with a description of the evaluation process in the next chapter.

Chapter 5

Evaluation

In the previous two chapters, the PACE system was introduced as a means of allowing private patient details to be shared between clinicians while keeping the bulk of patient data on the cloud. The query language used by the PACE system to query all relevant data sources was also detailed. We can now progress to evaluating the success of the PACE system as a potential solution to allowing clinics to use the cloud to store patient records while still enabling them to share confidential patient information. A prototype was developed to evaluate the PACE system and the experiments included functionality testing from nurses with real patient records shared between the users.

Section 5.1 gives an overview of the goals for the experiment and what questions the prototype should answer. In section 5.2, the prototype used to evaluate the PACE system is described. Section 5.3 describes the experiment used to test the functionality of the system and what results were collected. In section 5.4, we detail the experiment created to evaluate the performance of the system and what was learned from the results.

5.1 Overview

In order to evaluate the PACE system and answer the research questions posed in Chapter 1, there are two metrics that must be used when forming the experiments. The first metric is the functionality of the system: can the PACE system reliably store and retrieve patient data on the cloud and share private data between users. The second metric is the performance of the system: does the system perform well enough to allow clinicians to find and receive patient forms from other users at a speed that does not impede their treatment of patients. In measuring these metrics, we aim to answer our research questions.

In terms of the metric of functionality, we must look at what requirements the prototype fulfils. There are three main areas to concentrate on when evaluating this.

1. Using the prototype, can a user successfully interact with the cloud and store and retrieve patient medical records? The integrity of this data is vastly important, as missing data in these records can seriously affect the treatment of patients and destroy the trust in the application.

2. Can private patient details be effectively shared between users of the system? If peers are successfully sending data between each other, the next step in evaluating the P2P sharing is to ensure that the data being sent over is what was requested and uncorrupted.

3. Can the prototype build and propagate queries across the cloud and network that match patient instructions?

Once the prototype is demonstrated to achieve these targets, the real test for the functionality was deploying the prototype and allowing clinicians to test this functionality in a real-world setting with their own patient data.

Once the functionality of the PACE system prototype was tested and passed, the performance was then tested to ensure practicality. If finding and receiving patients

over a P2P network was too slow, the usability of the system would come under serious doubts, especially as a portable tool for clinicians interacting with patients and needing to reliably find their required information. In order to evaluate the performance of such a novel system, timings would be measured for operations.

5.2 Experimental Setup: PACE Prototype

In order to evaluate the PACE system, a prototype was built that was targeted at the requirements and research questions specified in Chapter 3. The prototype was built as part of the Elevator project [18] and aimed to provide the means for clinicians to upload patient details concerning their lifestyle and medical history relating to their mental health. Patient information is entered via a series of questionnaires through a web interface. The prototype was built with Googles App Engine cloud platform [23] and Java as the primary programming language. Using the design and architecture described in Chapter 3, the prototype uses a novel combination of both cloud and P2P technologies to facilitate managing patient records on the cloud and sharing their identifying values between registered users. Feedback and updates were given by the nurses involved with the Elevator project throughout the implementation of the prototype.

5.2.1 Cloud Implementation

The server side of the prototype was created using Google's App Engine and modelled on the architecture and design provided in Chapter 3. Below are descriptions of the key functionality implemented for the prototype.

User Authentication As mentioned in the requirements in Chapter 3, one of the requirements of the system is to ensure that only authorised clinicians are given access to the site. To accomplish this, Google's login service is utilised that requires every user to sign in with an email account before seeing the site. Once

the user signed in, they are asked to provide the password for their chosen clinic. This password is salted and hashed and compared to the password associated with the chosen clinic. If it matches, the user is given access to the site.

Servlets The PACE prototype used the MVP pattern, as described in the design, in order to manage the interaction between the user and the data. As the application was programmed with the Java programming language for the backend operations, the *view* is represented by Servlets. These collect input sent to it by the client application and can then be transformed into function calls to the backend services. A servlet was created for each page of the questionnaire to parse the input and two general servlets for operations such as signing in and out users, managing the peer network and adding new clinics and clinicians.

Data Management The medical records on the cloud are stored using Googles cloud oriented version of SQL, Cloud SQL. All data is stored using Hibernate, an Object-Relational Mapping (ORM) library for Java which maps object-oriented entities to relational databases [50]. Hibernate has the ability to take POJOs (Plain Old Java Objects) and translate it directly to a relational database. Thus, each patient, its medical details and the relationships between them were modelled as a series of POJOs, enabling us to use Hibernate to add and update patients with very simple commands. The only requirement for the objects to follow the necessary criteria was to ensure all the required variables were given `get` and `set` methods and annotated with keywords which help to define the generated tables and their relationships. To follow design best practices, a DAO (Data Access Object) was used to form an abstract interface between the operating classes of the system and Hibernate and the database. By keeping the patient data on Cloud SQL, the data can be viewed and tested using a standard RDBMS remote from the database.

When receiving the patient medical details over a HTTP connection, the data must be translated into a Java object in order to be stored by Hibernate. To do this, the Commons Bean Utils library from Apache was used which allows a Java Bean to be populated from a HTTP Request. This made the translation process

much simpler and cleaner as most pages of the questionnaire feature as many as 200 attributes. The Common Bean Utils library automated much of the task of generating the necessary object corresponding to the page of the form. Once the object was generated, it could be added to the corresponding `Patient` object and updated on the database. A similar problem was faced when operating the inverse procedure. When sending patient objects to the client, the Java object must be transformed into a web compatible format, namely a JSON (JavaScript Object Notation) object. To achieve this transformation, Googles GSON library [22] was used to transform the Java object into a Javascript compatible object that could be sent as a HTTP response.

Facilitating P2P Communication As part of the design of the PACE system, the cloud plays the part of the organising server to the hybrid decentralised topology used by the P2P network. In fulfilling the responsibilities of acting as the server, the cloud keeps a record of all signed in users as peers and also passes requests and instructions to the super-peers of the network. One of the biggest changes to the original design for the PACE system from this prototype is how super-peers are implemented on the system. As the user base for testing the prototype is rather small, the responsibility of the super-peer for each peer group is fulfilled by the cloud instead of an *elected* peer. As can be seen in figure 5.1, the topology is kept intact by the cloud. However, the functions of propagating requests and introducing peers are absorbed by the cloud application.

When a peer makes a request for a particular patient, the request is propagated across multiple peers. As several peers can potentially send the patient to the peer, a job system was implemented to prevent unnecessary P2P connections. When a peer makes a request for an individual patient, the P2P service on the cloud creates a `Job` object and sends the search criteria and `Job` identifier to the peers. If a peer has a patient matching the criteria, it makes a bid on the `Job` that matches the identifier it received. The first peer that makes a bid is sent the `Job` and introduced to the peer that requested the patient. If

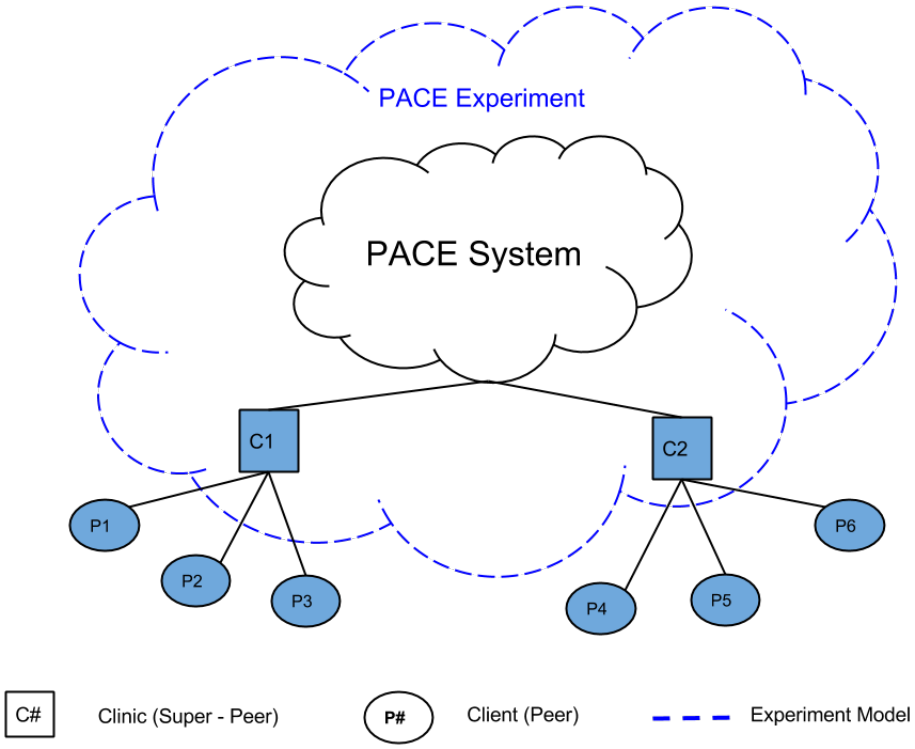

FIGURE 5.1: PACE Prototype P2P Topology

a peer replies to a `Job` that is already taken, they are informed that the `Job` no longer exists. Figure 5.2 demonstrates the sequence of steps necessary for finding and sending patient data over a P2P connection with the prototype. This job system ensures several peers are not sending the same data to a single peer, reducing the number of redundant P2P connections.

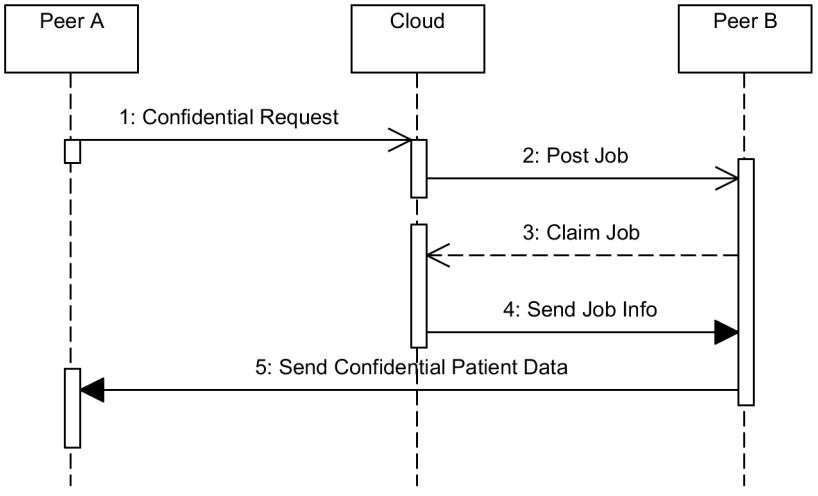

FIGURE 5.2: Retrieve Over P2P Network Sequence Diagram

5.2.2 Client Implementation

The client side of the prototype is a browser based application that targets Google's Chrome browser as it supports both the IndexedDB and WebRTC libraries. The following are descriptions for the functionality implemented for the client application.

User Interaction The interface for the PACE system prototypes client was created using standard web development tools: HTML5, CSS and JQuery. From the interface, the user can interact with the patient information on both the cloud and the private details shared among other users. The primary function for the interface, though, is to help the administration of a questionnaire to patients potentially suffering from degenerative cognitive illnesses such as dementia. This questionnaire consists of multiple pages relating to several areas of mental health (figures 5.4 and 5.3). As each page is completed, the page is submitted and added to the patients current record.

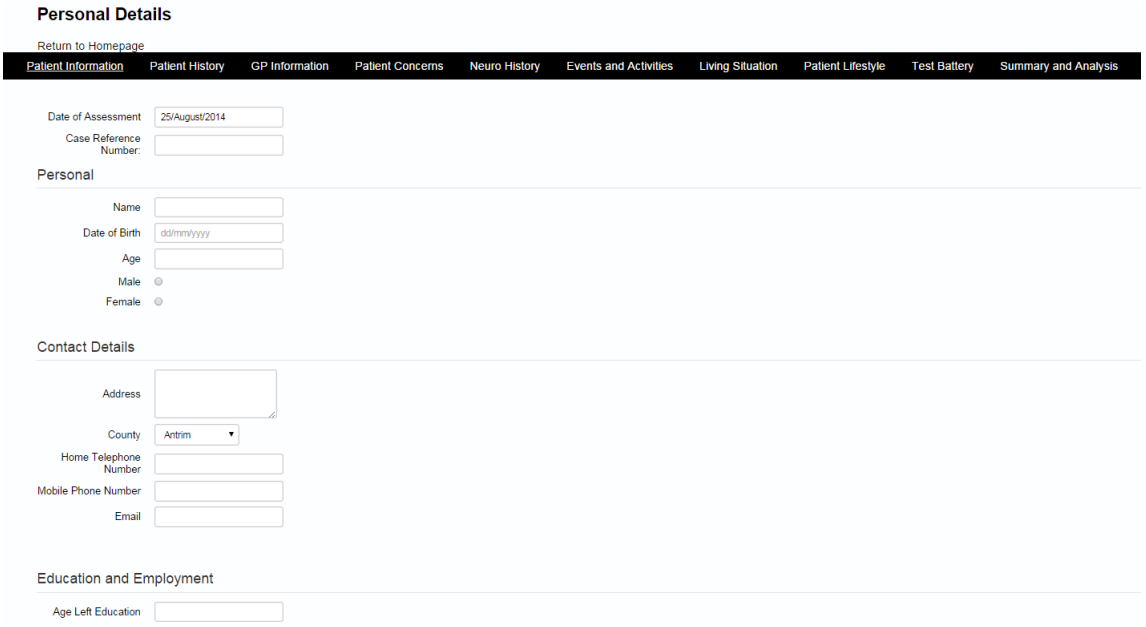

FIGURE 5.3: Questionnaire Page One - Includes Personal Details

Each patient added by the user can be reviewed and edited (figure 5.5).

The user can also use a name and date of birth to search for a patient on the system, which will lead to both confidential and anonymised data to be found

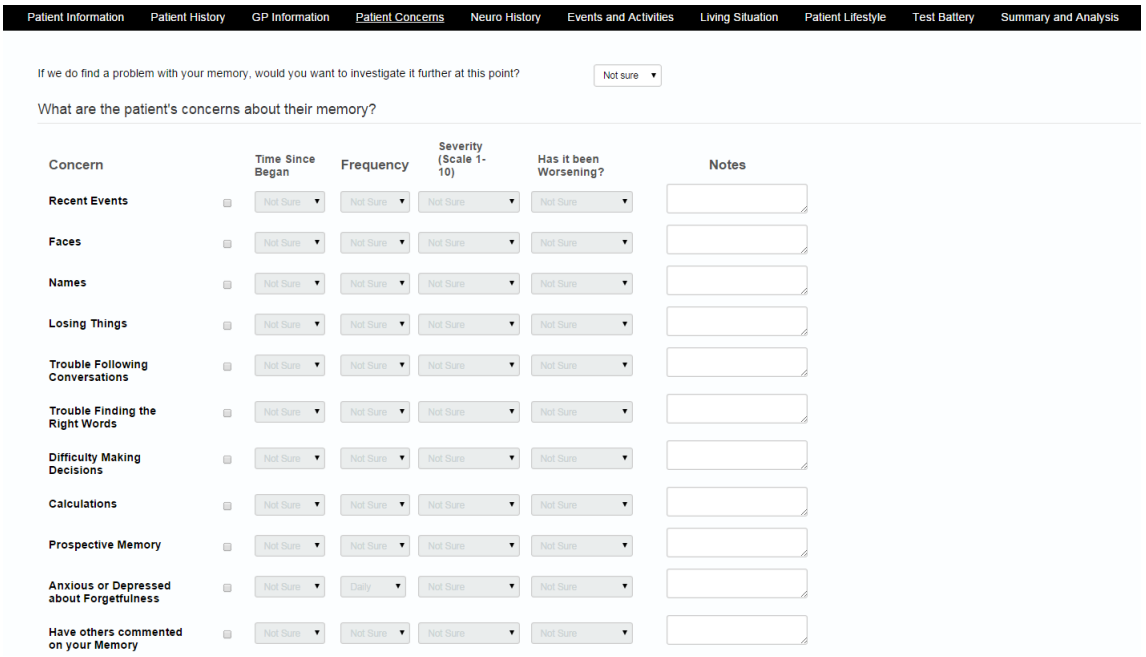

FIGURE 5.4: Questionnaire Page Three

FIGURE 5.5: Interface to Review or Delete a Patient

and combined before being presented to the user (figure 5.7). A patients entire medical history on the system can be reviewed through the interface (figure 5.6). From this review section, values can also be edited and then be used to update the information on the system.

Data Management The confidential data on the client machine is stored using IndexedDB; a browser-based database library that has been discussed previously in Chapter 3. The values stored locally are the name, date of birth, address, contact numbers, email, age, GPs (General Practitioner) name and GPs address. These are stored using a primary key generated using a one-way hash function. The generated primary key is then sent with the anonymised data to be made the primary key on the cloud as well. A DAO was also used for the

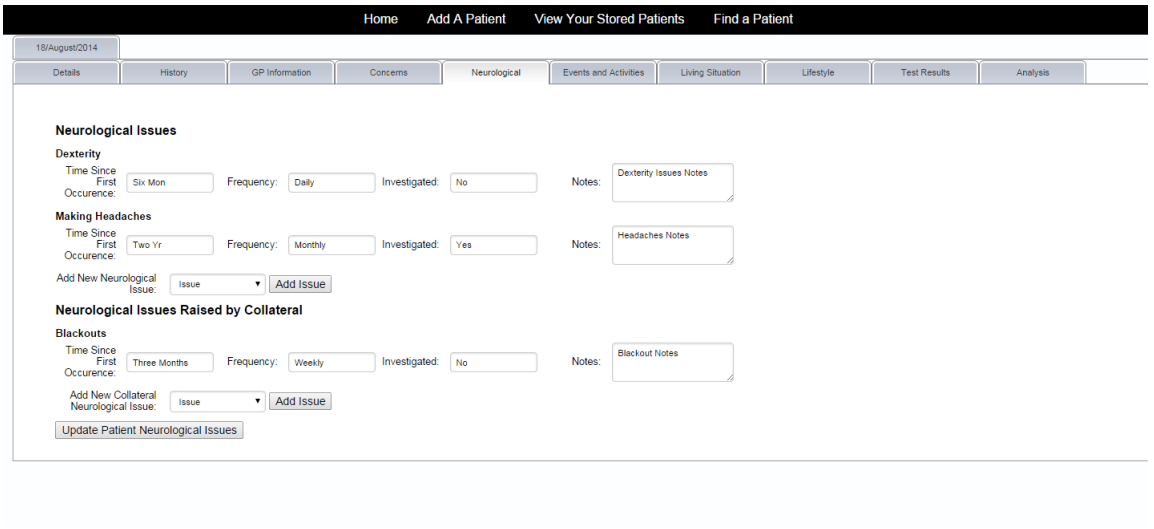

FIGURE 5.6: Interface for Reviewing Patient Details

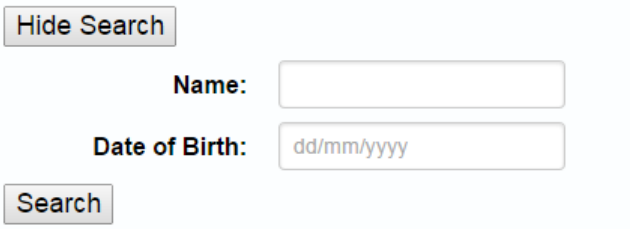

FIGURE 5.7: Interface to Search Patients with name and date of birth

client database to create an abstraction between the Javascript files and the confidential data.

P2P Communication The P2P functionality on the client is achieved using the WebRTC library. However, as the WebRTC project is an on-going project with frequent updates and changes, developing with the API alone can be difficult. To simplify the process of implementing WebRTC for the prototype, PeerJS[47] was used. PeerJS is a project aimed at making WebRTC communication easier and quicker to implement by providing a simplified API and library. As WebRTC has not currently been adopted by every browser, a backup strategy for enabling P2P communication was developed using Googles Channel API. If a peer requested patient data but was not using a supported browser, the patient data could be sent via the cloud. The data would first be packaged by the hosting peer into a JSON object, then sent directly to the cloud with a `Job` identifier (discussed previously in the chapter). When the cloud received the HTTP

message, it would transfer directly to the requesting peer via a Google Channel data channel. Although the private patient data is transferred across the cloud application, the time spent on cloud is minimal and never recorded and the strategy was required for users testing the prototype on various browsers.

5.2.3 Prototype Use Case

In order to demonstrate how the PACE system prototype operates, a use case will now be introduced and traced through the relevant processes until completion. In this scenario, the user requires to find the records of a patient for which they have no details stored locally on their machine except the name and date of birth of the required patient.

1. The clinician first enters the name and date of birth of the patient they require the information for, using the search interface shown in figure 5.7.

2. The entered information is converted into a patient identifier using the function `createID(patientName, patientDOB)`. This patient identifier is used to from a query that requests all the confidential columns of a patients entry in the PACE system database, as demonstrated in query 5.1.

```
SELECT ID Address, Contact_number, GP_address, GP_name, email
WHERE PID='22719472'
```

QUERY 5.1: Retrieve Confidential Data

3. This query is sent to the clinicians super-peer on the cloud. The super-peer creates a `Job` object that is stored on the system and the `Job` identifier is distributed to each peer currently online using Googles Channel API.

4. When a peer receives a job posting, the peer immediately sends a request for the job. The first request received by the cloud is accepted with all subsequent requests denied. The successful peer is then sent the `Job` object containing the full query and the address of the peer on the network.

5. The query is classified as a Retrieve operation on confidential information. The query is then transformed and the `getPatient(identifier)` function is used to locate the patient on their machine.

6. The successful peer can now send the required confidential data matching the patient identifier to the user using WebRTC. This is done by converting the data into a JSON object and using the peer address to form a connection with the PeerJS library.

7. Once the confidential data is retrieved, the information is displayed to the user to confirm it is the required patient. In order to see the patients records, the user simply pressed the Review button on the interface, which open the review page of the application, seen in the screen shot in figure 5.6.

8. As soon as this page is opened, the confidential fields are completed and the process of retrieving the first page of the form from the anonymised database begins. A new query is created which requests the first form from the patient with the matching identifier, as seen in query 5.2. In the query, the value 'Details' is the name of the first page on the patient's records.

```
SELECT Details
WHERE PID='22719472'
```

QUERY 5.2: Retrieve Anonymised Data

9. The cloud parses the query and extracts the patient identifier to first find the required patient using the `PatientDAO` function `getPatient(identifier)`. This returns a `Patient` object from which the latest record is found using the function `getLatestForm()`. Using the `Form` object, the Details page can be taken, converted to a JSON object using Gson and sent back to the user.

10. This JSON object is parsed by the browser and displayed on the review page for the user. These last two steps can be repeated for each page of the form.

The user has now retrieved the patient with the name and date of birth they specified and can review both the confidential and anonymised data retrieved from a peer and the cloud respectively. We now progress to evaluating the PACE system prototype.

5.3 Functionality Evaluation

There were two main areas to be focused on when evaluating the PACE system prototype, the functionality of the system and the performance. It was important firstly to know that such a novel system could operate correctly and match the requirements necessary for healthcare professionals.

5.3.1 Experiment

The goal for the functionality experiment was to evaluate the success of the PACE system prototype in fulfilling the requirements for a cloud-based patient data management system. The requirements include the four basic data operations Create, Retrieve, Update and Delete as well as being able to share all the data input onto the system between clinicians. In order to fully ensure that the system could perform the required operations, once the prototype was developed it was deployed on the cloud and nurses working as part of the Dementia ELEVATOR project used the system to retroactively input patient data to the system. Reviewing, editing and deleting patient data was also tested as part of the experiment.

In reality, we had only a limited number of people providing data. Thus, the nurses were placed in a single clinic to evaluate synchronising users of the same clinic which was prioritised over super-peer communication. Once multiple users were on the same clinic, sending and retrieving confidential data over P2P communications could be tested. The data received by a peer was then compared with the expected output to verify there was no corruption or missing values.

The nurses testing the system were given instructions to report any flaws when operating the system. These flaws could range from small bugs such as spelling that still allow the testing to pass to flaws that would result in the test declared a failure such as missing values or failure to operate. The experiment was run in three stages, each focusing on different operations. The experiment was conducted with the help of the nurses.

5.3.2 Results and Analysis

The results of the three functionality stages of the experiment are presented in tables 5.1-5.3 which show what operations were evaluated and if they failed, the reason was provided. Operations were implemented progressively to ensure that each operation performed successfully before proceeding to other operations that were dependent on the results of the previous stage. For example, dropped data could be misattributed to the `UPDATE` operation if the `CREATE` operation was not evaluated correctly. The `USER AUTHENTICATION`, `CREATE` and `RETRIEVE` operations were evaluated first, followed by the P2P operations, `SEND` and `RETRIEVE`. `UPDATE` and `DELETE` were the final operations to be evaluated.

5.3.2.1 Stage One

Operation	Success	Notes
USER AUTHENTICATION	✓	Users properly registered and unregistered users were given no access
CREATE	✗	Data dropped during creation
RETRIEVE	✗	Data dropped during retrieval

Table 5.1: Functionality Experiment Results: Stage One

The first experiment evaluated `USER AUTHENTICATION`, `CREATE` and `RETRIEVE` operations. These were the most important operations for the prototype as it allowed the users to enter patient data into the system and then review data to ensure there were no inconsistencies, all while being confident that the data could not be seen

by unregistered users. The `USER AUTHENTICATION` operation was made a priority before any data was created on the system but passed the testing phase allowing the experiment to continue. During this first stage, a number of small issues arose that resulted in the loss of data while being stored and also data not appearing on the review screen. This was due to the integration of heterogeneous technologies used to persist data on the cloud. The Apache library *Commons BeanUtils* was used to convert a HTTP request into a POJO (Plain Old Java Object) to store the data on the cloud using Hibernate. During this conversion process, a number of values were dropped leading to a `null` object being stored on the database. The same problem of dropped data was met while converting the POJO retrieved from the database to a JSON object using the GSON library.

5.3.2.2 Stage Two

Operation	Success	Notes
USER AUTHENTICATION	✓	Users properly registered and unregistered users were given no access
CREATE	✓	All data correctly stored on the cloud
RETRIEVE	✓	All data successfully retrieved from the cloud
SEND	✗	Could not communicate with other peers
RECEIVE	✗	Could not communicate with other peers

Table 5.2: Functionality Experiment Results: Stage Two

For experiment two, the `CREATE` operation passed the functionality test as no data was corrupted or lost. There were however inconsistencies in reviewing the data that resulted in the `RETRIEVE` function failing. The P2P communication was also implemented on the client side of the system, allowing the sending and receiving of data between peers to be tested. However, WebRTC did not seem to be able to communicate with other peers on the system effectively so no data could be sent or received. Thus, the `SEND` and `RECEIVE` operations for confidential data also failed. This was discovered to be an issue with PeerJS which was capable of making an initial connection between peers, but once the handshake protocol was completed,

the patient data could was never received. To bypass this issue for the next stage, the patient data was stored as part of the metadata in the initial connection between peers.

5.3.2.3 Stage Three

Operation	Success	Notes
USER AUTHENTICATION	✓	Users properly registered and unregistered users were given no access
CREATE	✓	All data correctly stored on the cloud
RETRIEVE	✓	All data successfully retrieved from the cloud
SEND	✓	Peers could successfully find and send data over P2P connection
RECEIVE	✓	Requested private data received from peer in full
UPDATE	✓	Patient Data successfully updated on cloud
DELETE	✓	Patient Completely removed from system

Table 5.3: Functionality Experiment Results: Stage Three

All operations were implemented for the third and final iteration of the experiment. At this stage, both the `RETRIEVE` and the P2P communications had been fixed and were working correctly. The `UPDATE` and `DELETE` operations were implemented for this experiment, allowing users to remove patients from the system and edit values through the review page. Both operations passed the user-testing at this stage. Thus, all operations passed on the third iteration of the experiment so we can now say that the PACE system prototype can successfully function in accordance with the requirements of the system users.

5.4 Performance Evaluation

Once the functionality of the prototype was evaluated, experiments were run to test the performance. As the system involved combining such novel technologies, it was

important to ensure that the system could perform quickly enough to justify the division of data which can often cause performance issues.

5.4.1 Experiment

For the experiment to measure the performance of the system, a key operation will be measured using a time metric. The operation that will be measured is the retrieval of data, both on the cloud and from peers. The reason only the retrieval will be measured is that is the only operation that involves the user waiting on the results, so any delay in retrieving the data would be easily noticed. The other operations are more asynchronous; the user can continue using the system while the changes are made to the patient data. The experiment used to evaluate the performance of data retrieval was done by measuring the time to search and retrieve data from the cloud and the peers on the system.

This experiment was run three times over three days to individual patients and a set of 100 patients. Each experiment involved retrieving the same data multiple times and calculating the average of the times. The average time of retrieval for each experiment, as well as the average of all the experiments, was recorded and compared between cloud and P2P retrievals. Although the timings for the P2P will not be identical to the cloud's metrics, there is a threshold for the metric that as long as the P2P communication stays within two seconds it is deemed acceptable for use. This pass/fail threshold was decided upon during discussions with the users and was considered to be the worst case scenario. The data being retrieved from the cloud is a single page of a record for each patient and the data being retrieved from a peer is all the confidential data corresponding to each patient.

5.4.2 Results and Analysis

Table 5.4 displays the recorded times for the three experiments measuring the performance of retrieving data for a single patient on the prototype. As shown the average

retrieval time for anonymised data from the cloud is 265 milliseconds. The time to retrieve confidential data from a peer is 1202 milliseconds and the average difference between the two is 937ms.

	Experiment 1	**Experiment 2**	**Experiment 3**	**Average**
Cloud	231ms	245ms	318ms	265ms
P2P	1182ms	1102ms	1322ms	1202ms
Difference	951ms	857ms	1004ms	937ms

TABLE 5.4: Single Patient Performance Experiment Results

Although the difference between P2P and cloud retrieval appears quite large and the average time for P2P retrieval is more than four times larger than the cloud equivalent however, the difference is still below the performance threshold of 2000ms, and so the effect should not be too adverse. There are also two main reasons why the P2P performs worse than the cloud, which are as a result of the implementation of the prototype rather than the design of the PACE system. The first reason is the job system that was implemented for the prototype, as discussed in section 5.2. As demonstrated in figure 5.2, there are 5 steps involved in acquiring data from another peer, whereas the cloud retrieval requires only two steps of communication: requesting the data and receiving the results. The job system was implemented only to request data from as many peers as possible from the network and will not be necessary on the inclusion of real super-peers.

This leads to the second reason for the performance of the P2P communications and that is the emulation of the super-peers. As the PACE system is a web application, nothing can be safely cached for quick access without the risk of deletion on the server. Thus, each step of the P2P retrieval process that involved the cloud requires multiple reads from the database, slowing down the performance. If super-peers where included, the read times should be much faster. This justifies our use of super-peers in our original design.

This reasoning is further evidenced when the results of retrieving multiple patients is analysed. Table 5.5 shows the average time for retrieving anonymous data for 100

patients is 488ms and the average time for confidential data retrieval is 1424ms. This is an average difference of 936ms. The average difference between the cloud and P2P for the two sets of experiments is approximately one millisecond.

	Experiment 1	**Experiment 2**	**Experiment 3**	**Average**
Cloud	413ms	501ms	552ms	488ms
P2P	1348ms	1467ms	1458ms	1424ms
Difference	935ms	966ms	906ms	936ms

TABLE 5.5: Multiple Patient Performance Experiment Results

This would suggest that the performance metrics of the cloud and P2P communications grow at the same rate as the amount of data increases and so the difference between the two remains a constant regardless of data size. Thus, by improving the P2P timings by including super-peers, as hypothesised, the P2P performance should remain comparable to the performance of the cloud for any reasonable amount of data.

5.5 Conclusions

In this chapter, we presented our evaluation of the PACE system, including a description of the prototype, the experiments that were run and the results that these experiments provided. The prototype that was implemented to evaluate the PACE system was created using Googles App Engine with Java. The prototype created had the capacity to allow clinicians to register as a user of the system, manage patient records and share data between users across a peer network. As the number of people using the prototype was quite small, super-peers were emulated on the cloud and a job system was introduced to allow peers to request information from all peers on the network.

Once the prototype was presented, we progressed to describing the series of experiments used to evaluate the functionality and the performance of the system. The

functionality experiment was run to ensure that the prototype matched the requirements of the users of the system. The experiment involved several nurses using the prototype to input historical patient information on the system and then share, review and edit the information. After three iterations of the experiment, all requirements were implemented and each operated successfully. The performance experiment involved measuring the time of retrieval operations in order to determine whether P2P communications would prove detrimental to a cloud application. The time to retrieve both one and one hundred patients was recorded several times and the averages and differences were compared. Although the cloud was faster to retrieve the data, the P2P communication was within the performance threshold and was also seen to maintain the same performance differential with the cloud regardless of the amount of data. Thus, the experiment showed that the inclusion of P2P communications does not significantly reduce the performance of a cloud application.

In this chapter we have presented a PACE system prototype, analysed results of both functional and performance experiments and successfully answered the research questions set out in chapter 1. In the next chapter, the work presented in this thesis is summarised and potential future work is outlined.

Chapter 6

Conclusions

The aim of this research was to design a system that allowed authorised users to manage patient data on the cloud and share confidential data between appropriate clinicians. Unlike other research projects, this work combined modern cloud and P2P technologies in a novel architecture that allowed the bulk of data to be stored on the cloud while the private patient details would be kept by the users of the system and shared via P2P communications. This allowed the elasticity and accessibility of the cloud to be leveraged while confidential data would exist on a P2P network capable of scaling up at the same rate as the cloud. As the data was divided and distributed over a network of users, a second objective for the research was the development of a system that allowed queries to be generated, spread across the network and cloud and the results of those queries to be combined in order to be presented to the user. In this chapter, an overall summary of the thesis is presented in section 6.1 and areas for future research are proposed in section 6.2.

6.1 Thesis Summary

In chapter 1, an overview of cloud computing and its place in the healthcare industry was presented. Healthcare facilities have sought to expand their IT operations onto the cloud in order to reduce the cost of managing the considerable load on site and to

facilitate the sharing of relevant patient information between different healthcare operations. The research presented in this thesis was conducted as part of the Dementia ELEVATOR project [18] which focuses on using modern technologies to analyse and improve the diagnosis and care for dementia patients. The problem the healthcare industry faces in using the cloud to manage patient data is the lack of trust associated with placing the confidential information in the hands of third party providers and current legislation governing the protection of patient information. Thus, the motivation for this research was designing a means of allowing healthcare operators to make use of the cloud while still allowing the sharing of confidential data between the users. Our hypothesis proposed that this problem could be solved by merging a cloud platform with modern P2P technologies in order to allow users to send private patient data directly to other authorised users. Our research questions focused on whether the real-world model of clinics and clinicians could be used to structure the network of peers and could SQLite-like queries be used to retrieve data spread across the cloud and peer network.

Several research projects were presented in chapter 2 aimed at protecting healthcare information on the cloud and supporting the sharing of patient data between different users. We focused on three different categories of research; the first was protecting confidential data on the cloud through encryption based strategies. The current *state-of-the-art* strategies for encrypting data seems to focus on *attribute-based encryption* making it possible to perform queries on the encrypted data. However, the performance cost and overhead associated with encryption and preparing the data for queries made it an inefficient solution. The second category of research that was investigated was the role of P2P technologies in sharing confidential data in the industry of healthcare. The mentioned research projects implemented the P2P networks using JXTA and included solutions for sharing data within a single hospital and across multiple facilities. The solutions presented relied heavily on peers having the capacity to store a lot of data and extraneous hardware which could be unsuitable for clinics that do not have the finances to implement them. The research projects

presented for the final category were those that expanded on standard cloud infrastructures by introducing multiple cloud structures or P2P networks. Although both solutions improved upon the singular cloud architecture, neither solution presented took advantage effectively of combining the two technologies.

In chapter 3, our PACE (P2P Architecture for Cloud-based EHealth) system was introduced. The PACE system is a novel combination of cloud and P2P technologies designed to answer the research questions presented in chapter 1. The cloud portion of the PACE system was designed using the MVP (Model View Presenter) pattern which was described as helping to form a level of abstraction between the user interface and the systems data. The system stored two types of data, the administrative data such as the information on the clinicians (users) and the clinics and the patient data, which included the anonymised medical records. The design for the P2P network was then introduced, which mirrored the model used for clinics and clinicians. Each clinic was represented as a super-peer which would manage the peers (clinicians) on the network. Each peer on the network holds a store of confidential patient information that is linked with the anonymised information on the cloud using a shared primary key. This confidential data could then be shared with other peers via a P2P communication. The focus for this design was the sharing of all patient information on the system and being able to locate and recombine the anonymised and confidential data for the user.

The architecture that underpinned the PACE system was then described. This provided details of the components necessary on the server end of the system, based on the cloud, and the client application which would be run on an internet browser application. The four main components for the server architecture were described as the *Client Interface*, the *Clinic Connector*, the *Peer Data Access* and the *Patient Data Access*. The components for the client were then introduced as the *Client Application*, the *Peer Connector* and the *Private Patient Data Access*. The responsibilities and relationships of each of these components were detailed and their place in the architecture was justified throughout sections 3.4 and 3.5. We then progressed to describing the various technologies that were researched that would need to be utilised

to implement the PACE system architecture. Google App Engine was chosen as the preferred cloud provider due to its advanced client communication capabilities and low cost. WebRTC was then introduced as a means for allowing web applications on a browser to establish P2P connections in order to send private patient information. This was followed by a summary of the various approaches to storing this patient data on a browser and IndexedDB was outlined as the most suitable library.

One of the major challenges confronted while designing the PACE system was how it would approach retrieving data that had been divided and distributed across multiple sources. In chapter 4, we describe how queries would be used in the PACE system in order to locate and retrieve patient data. As part of the design, when a user seeks to retrieve or alter the data stored anywhere on the system, a query would be generated that could be delivered to the various data locations and the operation performed. The chapter began with the requirements for how the query system should perform and included the functionality that would need to be supported and the expectations of how the data should be treated. We then progressed to defining how a query should be classified using three categories: the level of confidentiality of the data the query would affect, the scope of the query, or in other words whether the query would be parsed at the peer, super-peer or cloud level and the final category is the operation of the query. The four operations that should be supported by the query system are *Create*, *Retrieve*, *Update* and *Delete*. This classification step is necessary to improve the aid the different components of the PACE system in deciding how a query should be parsed and where it should be sent. The lifecycle of the query was then outlined and included descriptions of how the queries are constructed, how they are parsed and the functions that are called as the query is transformed.

The evaluation of the PACE system was then detailed in chapter 5. The functionality and performance of the system were the focus of the evaluation to ensure that the PACE system matched the requirements set out by the nurses that would be using the system. The chapter began with a description of the prototype used to evaluate the system. This description included details of how the cloud and client were implemented and the differences that existed between it and the original design. The

functionality evaluation was then outlined and involved ensuring the system could perform the necessary operations and were conducted over three stages, with each stage focusing on new operations to be evaluated by the users. The results for the functionality experiment were seen to yield positive results as all necessary operations performed as expected. Lastly, the experiment to evaluate the performance of the system was defined. As described in section 5.4, the evaluation was split into two sets of three experiments. The time to retrieve a single patient from the cloud and over the P2P network was measured three different times and an average was calculated. These times were then compared to the time taken to retrieve 100 patients from the cloud and the peer network. An analysis of these results was then provided and it was discovered that the performance of the P2P was comparable with the cloud. Thus, the PACE system presents a realistic solution to problem of sharing patient data on the cloud using P2P.

6.2 Future Research

Over recent years, cloud computing has become almost eponymous with sharing and storing information and media over the internet. The platform has presented itself as capable of helping the healthcare industry take a large meaningful step forward in managing patient's medical records. Unfortunately, cloud computing providers have not proved that it can deliver the level of trust necessary for healthcare professionals to entrust such a third party with keeping a patient's personal details safe and secure. Although many researchers have attempted to produce a system that can provide the necessary security, none have produced a system capable of convincing the healthcare industry. The PACE system has addressed these security concerns by extracting identifiable values from cloud-stored records but still provide a way for users to share complete records with others. However, there is still more research that may be conducted to improve upon the PACE system's limitations. The final section of this thesis will describe directions to be taken to further this research.

6.2.1 Maintaining Data Consistency between Peers

One limitation of the PACE system that could be improved upon is the method of enforcing consistency of patient data between peers of the same peer group. The current system involves peers contacting the super-peer and informing it of updated confidential data. The super-peer will then inform each other peer that information has been updated and to contact the peer with the change. This sequence of steps involves the peer with original update transferring the same data multiple times and the time to share all this data is of the magnitude of `O(n)`. By introducing a more efficient protocol for updating the time to update the group could be substantially improved. For instance, a chord-based gossip protocol in which each peer shares any update query it receives with its *neighbours* could potentially help to ensure that data is consistent across the peer group.

6.2.2 Expanding the System for More Areas of Healthcare

Currently, this research has focused on managing and sharing patient records with an emphasis on dementia related care. However, we believe there is scope for the PACE system to be expanded to accommodate a larger subset of the patient data generated through the healthcare industry. In particular, by introducing domains to distinguish between different areas of the medical profession (mental health, physical injuries, chronic diseases etc.), the system could automatically retrieve relevant information from the patient's medical records dependent on the speciality of the user. The set of private values for the patient should remain mostly consistent across all departments of healthcare, while the cloud can sustain the growth of adding new domains. The system already supports multiple pages of values for each medical record for a patient. Thus, a similar approach could be adopted by allowing multiple different records for each profile and each record would correspond to a different medical domain. This would have the benefit of allowing multiple healthcare departments and facilities to share and collaborate in the diagnosis and treatment of the patients.

Bibliography

[1] Sanjay P Ahuja, Sindhu Mani, and Jesus Zambrano. A Survey of the State of Cloud Computing in Healthcare. *Network & Communication Technologies*, 1(2):12–19, 2012.

[2] Amazon. Amazon web services: Overview of Security Processes. Technical report, Amazon, 2014.

[3] Amazon. Amazon Web Services Price Calculator, 2014.

[4] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A Survey of Peer-to-Peer Content Distribution Technologies. *ACM Computer Surveys*, 36(4):335–371, December 2004.

[5] Zohra Bellahsene and Mark Roantree. Querying Distributed Data in a Super-Peer Based Architecture. In *Database and Expert Systems Applications*, volume 3180 of *Lecture Notes in Computer Science*, pages 296–305. Springer, 2004.

[6] B Beverly Yang and Hector Garcia-Molina. Designing a Super-peer Network. In *Data Engineering, 2003. Proceedings. 19th International Conference on*, pages 49–60. IEEE, 2003.

[7] Sushil Bhardwaj, Leena Jain, and Sandeep Jain. Cloud computing: A study of infrastructure as a service (IAAS). *International Journal of engineering and information Technology*, 2(1):60–63, 2010.

[8] Jean-Paul Boodhoo. Design patterns: Model view presenter. Technical report, Microsoft, August 2006. http://http://msdn.microsoft.com/en-us/magazine/cc188690.aspx.ie.

[9] Michelle Butler, Margaret Treacy, Anne Scott, Abbey Hyde, Pádraig Mac Neela, Kate Irving, Anne Byrne, and Jonathan Drennan. Towards a Nursing Minimum Data Set for Ireland: Making Irish Nursing visible. *Journal of Advanced Nursing*, 55(3):364–375, 2006.

[10] Mario Cannataro, Domenico Talia, Giuseppe Tradigo, Paolo Trunfio, and Pierangelo Veltri. SIGMCC: A System for Sharing Meta Patient Records in a Peer-to-Peer Environment. *Future Generation Computer Systems*, 24(3):222–234, 2008.

[11] Lingfeng Chen and Doan B Hoang. Novel Data Protection Model in Healthcare Cloud. In *Proc. of IEEE International Conference on High Performance Computing and Communications*, pages 550–555, 2011.

[12] Yu-Yi Chen, Jun-Chao Lu, and Jinn-Ke Jan. A Secure EHR System Based on Hybrid Clouds. *Journal of Medical Systems*, 36(5):3375–3384, 2012.

[13] B. Coats and S. Acharya. Bridging Electronic Health Record Access to the Cloud. In *Proc. of 47th Hawaii International Conference on System Science*, pages 2948–2957. IEEE, 2014.

[14] A. Dogac, G.B. Laleci, S. Kirbas, Y. Kabak, S.S. Sinir, A. Yildiz, and Y. Gurcan. Artemis: Deploying Semantically Enriched Web Services in the Health Domain. *Information Systems*, 31:321–339, 2006.

[15] Neil Donnelly, Kate Irving, and Mark Roantree. Cooperation Across Multiple Healthcare Clinics on the Cloud. In *Distributed Applications and Interoperable Systems*, Lecture Notes in Computer Science, pages 82–88. Springer, 2014.

[16] Jonathan Drennan, Margaret P Treacy, Michelle Butler, Anne Byrne, Gerard Fealy, Kate Frazer, and Kate Irving. Support Networks of Older People Living in the Community. *International Journal of Older People Nursing*, 3(4):234–242, 2008.

[17] C Eikemeier. Introducing P2P in Healthcare. *Swiss Medical Informatics*, 51:6–9, 2003.

[18] Dementia Elevator. Elevator: Building Dementia Skills Capacity, 2014. http://dementiaelevator.ie.

[19] Benjamin Fabian, Tatiana Ermakova, and Philipp Junghanns. Collaborative and Secure Sharing of Healthcare Data in Multi-Clouds. *Information Systems*, 2014.

[20] Borko Furht and Armando Escalante. *Handbook of Cloud Computing*. Computer science. Springer, 2010.

[21] Li Gong. Jxta: A Network Programming Environment. *Internet Computing, IEEE*, 5(3):88–95, 2001.

[22] Google. Gson, April 2011. https://sites.google.com/site/gson/.

[23] Google. Google App Engine, June 2014. https://appengine.google.com.

[24] Google. Google Cloud Price Calculator, 2014. https://cloud.google.com/products/calculator/.

[25] Google. Google's Approach to IT Security. Technical report, 2014. https://cloud.google.com/files/Google-CommonSecurity-WhitePaper-v1.4.pdf.

[26] Vipul Goyal, Omkant Pandey, Amit Sahai, and Brent Waters. Attribute-Based Encryption for Fine-Grained Access Control of Encrypted data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, pages 89–98. ACM, 2006.

[27] Yang Guo, Yan Hu, Javed Afzal, and Guohua Bai. Using P2P Technology to Achieve eHealth Interoperability. In *2011 8th International Conference on Service Systems and Service Management (ICSSSM)*, pages 1–5. IEEE, 2011.

[28] Kate Irving, Paulina Piasek, Sophia Kilcullen, Ann-Marie Coen, and Mary Manning. National Educational Needs Analysis Report. Technical report, Dublin City Univeristy and Heatlhcare Service Executive, 2014. http://dementiaelevator.ie/wp-content/uploads/2013/12/Elevator-National-Educational-Needs-Analysis-Report-Print-Version.pdf.

[29] Sandra Kalmijn, Lenore J Launer, Alewijn Ott, Jacqueline Witteman, Albert Hofman, and Monique Breteler. Dietary Fat Intake and the Risk of Incident Dementia in the Rotterdam Study. *Annals of Neurology*, 42(5):776–782, 1997.

[30] Alex Mu-Hsing Kuo. Opportunities and Challenges of Cloud Computing to Improve Health Care Services. *Journal of Medical Internet Research*, 13(3):3, 2011.

[31] Sai Ho Kwok, KY Chan, and YM Cheung. A server-Mediated Peer-to-Peer System. *ACM SIGecom Exchanges*, 5(3):38–47, April 2005.

[32] LJ Launer, K Andersen, MEea Dewey, L Letenneur, A Ott, LA Amaducci, C Brayne, JRM Copeland, J-F Dartigues, P Kragh-Sorensen, et al. Rates and Risk Factors for Dementia and Alzheimer's Disease Results from EURODEM Pooled Analyses. *Neurology*, 52(1):78–84, 1999.

[33] Avraham Leff and James T Rayfield. Web-Application Development Using the Model/View/Controller Design Pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*, pages 118–127. IEEE, 2001.

[34] Ming Li, Shucheng Yu, Ning Cao, and Wenjing Lou. Authorized Private Keyword Search Over Encrypted Data in Cloud Computing. In *Distributed Computing Systems (ICDCS), 2011 31st International Conference on*, pages 383–392. IEEE, 2011.

[35] Ming Li, Shucheng Yu, Kui Ren, and Wenjing Lou. Securing Personal Health Records in Cloud Computing: Patient-centric and Fine-Grained Data Access Control in Multi-Owner Settings. In *Security and Privacy in Communication Networks*, pages 89–106. Springer, 2010.

[36] Ming Li, Shucheng Yu, Yao Zheng, Kui Ren, and Wenjing Lou. Scalable and Secure Sharing of Personal Health Records in Cloud Computing Using Attribute-Based Encryption. *Parallel and Distributed Systems, IEEE Transactions on*, 24(1):131–143, 2013.

[37] Byong-In Lim, Kee-Hyun Choi, and Dong-Ryeol Shin. A Secure Peer-to-Peer Group Collaboration Scheme for Healthcare System. In *Computational Science–ICCS 2005*, pages 346–349. Springer, 2005.

[38] Byongin Lim, Keehyun Choi, and Dongryeol Shin. A jxta-based Architecture for Efficient and Adaptive Healthcare Services. In *Proc. of International Conference on Information Networking. Convergence in Broadband and Mobile Networking*, pages 776–785, 2005.

[39] Hans Löhr, Ahmad-Reza Sadeghi, and Marcel Winandy. Securing the E-Health Cloud. In *IHI'10 Proc. of the ACM International Health Informatics Symposium*, 2010.

[40] Prodromos Makris, Dimitrios N Skoutas, Panagiotis Rizomiliotis, and Charalabos Skianis. A User-Oriented, Customizable Infrastructure Sharing Approach for Hybrid Cloud Computing Environments. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 432–439. IEEE, 2011.

[41] Michael McKeown, Hanu Kommalapati, and Jason Roth. Disaster Recovery and High Availability for Azure Applications. Technical report, Microsoft, 2014.

[42] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, 53(6):50, 2009.

[43] Bret Michael. In Clouds Shall We Trust? *IEEE Security & Privacy*, 7(5):3, 2009.

[44] Microsoft. Microsoft Azure Price Calculator, 2014.

[45] Mozilla. IndexedDB, 2014. https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API.

[46] Tatsuaki Okamoto and Katsuyuki Takashima. Hierarchical Predicate Encryption for Inner-Products. In *Advances in Cryptology–ASIACRYPT 2009*, pages 214–231. Springer, 2009.

[47] PeerJS. PeerJS. http://www.peerjs.com.

[48] Martin Prince, Maëlenn Guerchet, and Matthew Prina. Policy Brief for Heads of Government: The Global Impact of Dementia 2013-2050, 2013. Last accessed 15.05.14.

[49] Rajasekaran Rajkumar and Nallani Chackravatula Sriman Narayana Iyengar. Dynamic Integration of Mobile JXTA with Cloud Computing for Emergency Rural Public Health Care. *Osong Public Health and Research Perspectives*, 4(5):255–264, 2013.

[50] Inc. Red Hat. Hibernate, April 2014. http://www.hibernate.org.

[51] Mark Roantree, Jie Shi, Paolo Cappellari, Martin F O'Connor, Michael Whelan, and Niall Moyna. Data Transformation and Query Management in Personal Health Sensor Networks. *Journal of Network and Computer Applications*, 35(4):1191–1202, 2012.

[52] C.O. Rolim, F.L. Koch, C.B. Westphall, J. Werner, A. Fracalossi, and G.S. Salvador. A Cloud Computing Solution for Patient's Data Collection in Health Care Institutions. In *Proc. of Second International Conference on eHealth, Telemedicine and Social Medicine*, 2010.

[53] David E.Y. Sarna. *Implementing and Developing Cloud Computing Applications*. An Auerback book. CRC Press, 2011.

[54] Stelios Sotiriadis, Euripides GM Petrakis, Stefan Covaci, Paolo Zampognaro, Eleni Georga, and Christoph Thuemmler. An Architecture for Designing Future Internet (FI) Applications in Sensitive Domains: Expressing the Software to Data Paradigm by Utilizing Hybrid Cloud Technology. In *13th IEEE International Conference on BioInformatics and BioEngineering*. IEEE, 2013.

[55] Ton Spil and Richard Klein. Personal Health Records Success; Why Google Health Failed and What Does that Mean for Microsoft Health Vault. In *Proc. of 47th Hawaii International Conference on System Science*, pages 2818–2827, 2014.

[56] L Taylor. Spring Security in Google App Engine, August 2010.

[57] Irena Trajkovska, Joaquin Salvachua, and Alberto Mozo Velasco. A Novel P2P and Cloud Computing Hybrid Architecture for Multimedia Streaming with QoS Cost Functions. In *MM '10 Proc. of Internation Conference on Multimedia*, 2010.

[58] Mitch Tulloch. *Introducing Windows Azure for IT Professionals*. Introducing. Pearson Education, 2013.

[59] J Vairia and S. Mathew. *Overview of Amazon Web Services*. Amazon, https://d36cz9buwru1tt.cloudfront.net/AWS_Overview.pdf, January 2014.

[60] Christian Vogt, Max Jonas Werner, and Thomas C Schmidt. Leveraging WebRTC for P2P Content Distribution in Web Browsers. In *21st IEEE Intern. Conf. on Network Protocols (ICNP 2013), Demo Session. Piscataway, NJ, USA: IEEEPress*, 2013.

[61] W3C. Web SQL Database, 2010. http://www.w3.org/TR/webdatabase/.

[62] W3C. Web Storage, 2013. http://www.w3.org/TR/webstorage/.

[63] W3C. WebRTC 1.0: Real-Time Communication Between Browsers, 2013. http://www.w3.org/TR/webrtc/.