A DISSERTATION
ON

AUGMENTATION IN UCON MODEL

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE AWARD OF THE DEGREE OF MASTER OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING

SUBMITTED BY:
NIDHI JOSHI
2K13/CSE/27

UNDER THE GUIDANCE OF:
MR. MANOJ KUMAR
(ASSOCIATE PROFESSOR)

DEPARTMENT OF COMPUTER ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
JUNE 2016

A
Dissertation
On

Augmentation in UCON Model

Submitted in Partial Fulfillment of the Requirement
For the Award of the Degree of

**Master of Technology
In
Computer Science & Engineering**

Submitted By:
**Nidhi Joshi
2K13/CSE/27**

Under the Esteemed Guidance of:
**Mr. Manoj Kumar
(Associate Professor)**

**DEPARTMENT OF COMPUTER ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
JUNE 2016**

ABSTRACT

In this dissertation, we have presented a new architecture of UCON access control model for e-healthcare domain. This enhanced UCON model can scale effectively as per user demand, perform isolation of sensitive data from non-sensitive data and data transmission between different modules is via secure model and thus helps in achieving high availability, performance, scalability, resilience and security. Features of Cloud computing have also been explored and used to achieve some of the features listed above. Existing implementation of UCON for e-healthcare domain do not cater for all of these requirements, for an e-healthcare system it is imperative to have an access control model which fulfills all of these.

New architecture comprises of a client application, user platform on local network and policy server on cloud platform. Sensitive and Non-sensitive data has been segregated to maintain data isolation. Sensitive data is kept on local network of hospital and non-sensitive data on cloud. Multiple UCON policy servers are available on cloud to cope with component failures and to provide resilience.

Data communication from user platform to cloud is through SSL to maintain data security. Feature of Sun XACML has been used in the implementation of policy server. All features of UCON model related to authorization, condition, obligation, continuity and attribute mutability has been incorporated in the proposed model.

ACKNOWLEDGEMENT

First and foremost I would like to thank the **Lord Almighty** for showering his blessing in all endeavours.

With immense pleasure I take this opportunity to express my indebtedness gratitude to our beloved Vice chancellor **Prof. Yogesh Singh** who is enriching keen interest in academic pursuits.

I convey my sincere thanks to our HOD **Prof. O.P. Verma,** Department of CSE for his kind encouragement and motivation to complete this Project successfully.

I profoundly thank our respected Associate Professor **Mr. Manoj Kumar** Department of CSE, for his full-fledged support and guidance throughout the Project.

Last but not least I render my heartiest complements to all my **Staff Members, Librarian, Family and Friends** for giving their valuable suggestions, encouragement and support for completing my project successfully.

NIDHI JOSHI
University Roll no.: 2K13/CSE/27
M. Tech (Computer Science & Engineering)
Department of Computer Engineering

Department of Computer Engineering
DELHI TECHNOLOGICAL UNIVERSITY
Shahabad Daulatpur, Main Bawana Road,
Delhi-110042

CERTIFICATE

This is to Certify that the dissertation titled **"Augmentation in UCON Model "** is a bonafide record of work done by **NIDHI JOSHI, ROLL NO.: 2K13/CSE/27 at Delhi Technological University** for partial fulfilment of the requirement for the degree of Master of Technology in Computer Science & Engineering. This project was carried out under my supervision and has not been submitted elsewhere, either in part or full, for the award of any other degree or diploma to the best of my knowledge and belief.

Sh. Manoj Kumar
Date: ____________
Associate Professor & Project Guide
Department of Computer Engineering
Delhi Technological University

Table of Contents

Chapter 1
Introduction

Nowadays, computing environment is dynamically changing which demands for new technical approaches to cope with the increasing security challenges. Access control is one of the earliest computer securities and it still remains a continuing challenge. There are many traditional access control models available but some of them uses static authorization decisions which implies subjects have pre-assigned permission on objects, so it is more of a closed system and hence not suitable in a dynamic environment. In very large distributed open systems such identity based models do not work because relationship between subject and object is dynamic. System demands usage of attributes or characteristics in access control models. There are many models like ABAC, RABAC developed on this concept but each one has its own challenges. Distributed system requires an access control model which excels in following parameters:

1. User’s convenience – access mechanism should be transparent to the user and access decision making model should be faster and have low latency
2. Performance & Scalability – Access control model should be able to scale up easily and sustainably with increase in number of operations or number of users. With an increase in aforementioned parameters (operations/users) system should exhibit minimum deviation from the average time to respond
3. Security – Sensitive user data should remain confidential and protected against unwanted threats. This can be achieved by using some cryptographic methods in the Access Control model used. Cryptographic methods chosen should ensure less overhead in key distribution and data management
4. Attribute Mutability and Continuity – Continuity caters to continuous access re-evaluation during the course of usage and mutability ensures any changes that may impact access decision are enforced in the same window. These two features are imperative in a dynamic computing environment

1.1 Fundamentals of Access Control System

Access Control is a technique to control who or what can use or view resources in a computing world. It limits access to a system or to a virtual or to a physical resource. Access control mainly comprises of two entities – subject and object. Subject is an entity making the request; it can be a user or a program or a process operating on behalf of the user. Subject interacts with the system and accesses the desired resources. Objects are resources which subject wishes to access; it could be anything like mp3 file, image, document etc. Access control system grants or revokes the access to the object based on the information provided and authorization policies defined.

1.2 Types of Access Control Model

Some of the access control models have been listed below:

1. **Discretionary Access Control –** In this model, identity of subject is used to control the access. Owner of object decides its access permission for other users and accordingly sets them. Hence, it provides flexibility of usage on information. Example of such model is the UNIX operating system - the subject/user can specify what permissions (read/write/execute) members in the same group may have and also what permissions all others may have.
   In such systems, unauthorized users can easily access the information because there is no control on copies of resources and no control on flow of information. Consistency of information is hampered in this model. These models are typically used only with legacy applications and will incur considerable management overhead in the modern multi-user and multi-application environment.

2. **Mandatory Access Control –** In this model, access decisions are controlled by the attaching security level on subject and object. This helps in solving the information flow problem. All users need to obtain certain clearance to access objects, a relationship should exist between two. An individual doesn’t have the authority to change the access, the access permissions are decided by the administrator of the system, and not by the subject. Mainly MAC takes hierarchical approach to control the access; hierarchy depends on the security levels e.g., Unclassified, Classified, Secret and Top Secret. Security labels are assigned to all subjects based on the object they request for. Labels propagate to derivative objects, including their copies. Information integrity is increased as information flow from lower level to higher level. It is mostly used in military and government applications where multilevel security is required. Drawback in MAC is that once the security level is defined for a particular subject in the hierarchy it doesn’t modify it. MAC and DAC are fixed access control models as their policies are hard to change.

3. **Role Based Access Control –** In this model, access decisions are made based on the roles and responsibilities of subject. Roles are defined based on the job functions; it can be set of objects or actions that are associated with a subject. It can be managed centrally. Permissions are defined based on job authorities and responsibilities of the subject. Actions/Operations on an object are driven by the permissions assigned to the subject. This model is more scalable than DAC and MAC models and is suitable for cloud computing environment where relationship between users and services is dynamic. Roles can be assigned based on least privilege to minimize the risk of intrusion. At times it is hard to reach which privilege is associated with which user for a particular role. Permissions associated with each role can be deleted or modified based on the privilege of role change. It doesn't take into consideration contextual information while making access decision, in order to achieve that more roles might need to be created and creating excessive number of roles creates a problem called role explosion.

4. **Attribute Based Access Control –** In this model, access control is determined based on the attributes of the entities involved. For example: department is a user attribute, createTime and size are object attributes. Fine grained access policies which is required by most of the applications can be easily defined with the help of ABAC. Role based access control has a problem of assigning privileges to the subject; this is solved by ABAC as it uses a set of subject's attributes. ABAC allows more comprehensive rules/policies to be formed as compared to traditional models. In ABAC, once the authorization policies are defined, authorization is computed at the time of request and there is no need for pre-assignment of permissions to the users. This model is more flexible and scalable in comparison to the models stated above. However, using an organizational hierarchical data it is difficult to implement ABAC model. Its major challenge is just-in-time policy evaluation which does not factor in change in condition of attributes which require re-evaluation of authorization decision. Hence, disconnect between the rule based policies and the resources.

5. **Usage Control Model –** It is the next generation access control model because of its unique property of decision continuity. It is an attribute based model, thus permissions are based on attributes of subjects, object or environments and policies defined in the form of authorization, condition and obligation. Two vital properties of UCON are attribute mutability and continuity. Change in subject's or object's attribute as a side effect of usage comes under attribute mutability and continuity refers to repeatedly checking the validity of subject's right

on the object. Authorization may lead to performing update on subject or object attribute. To enforce mutability, there can be pre-updates, post updates or on-going updates. For continuity properties, there can be three parts – on-going usage, before and after usage. Based on these, different cases of UCON models can exist.

1.3 Introduction to Cloud Computing

Cloud Computing is a computing model which provides infrastructure, applications and platform as per pay-per-user cost model. Most of the organizations are moving towards using cloud due to the numerous capabilities and features provided by it. There are three main models in cloud computing:

1. **Software as a service (SaaS) –** In this cloud providers enables users to access the applications hosted on cloud using web browser. Users do not need to install the application on their local machine
2. **Infrastructure as a service (IaaS) –** In this cloud providers provide resources like storage, network, processing etc to install and run the applications and operating system
3. **Platform as a service –** In this cloud providers provide platforms to develop applications. Client can create their applications using cloud platform and the languages provided by cloud

With the increase in usage of cloud, the attacks on cloud computing has also been increased. Access control is one of vital security solution for the data in cloud. There are different types of cloud available, based on an individual’s or organization requirement the type of cloud to be used can be chosen. Some of the clouds available are:

1. **Public Cloud –** All clients on public cloud share same infrastructure pool with limited security protections and configuration options. It is beneficial when you need to test and develop an application code, when you are doing a collaboration project. Some of public cloud providers are Amazon , Google and Microsoft
2. **Private Cloud –** Such clouds are dedicated to an organization and allow them to host their applications on cloud taking into consideration data security and control. Such clouds are not shared by other clients. They are further classified into two types:
    i. On Premise Private Cloud: Cloud is hosted inside the organization and it would incur the capital and operational cost of the physical resources in this model.
    ii. Externally Hosted Private Cloud: Cloud is hosted by a third party which ensure full guarantee of privacy

   It is generally opted by organization where data control is crucial or organizations want to set up a data centre of their own

3. **Hybrid Cloud –** It is formed by combining two or more types of cloud. In this type, client can leverage the cloud provider in either partial or full manner for eg: secured data can be kept in on premise private cloud and application can be hosted in public cloud.
4. **Community Cloud –** It is a variation of private cloud which is built for targeted group. It is governed, managed and secured either by participating organizations or by the cloud service provider. It can be used by organizations like a state government bodies that need to share the resources among themselves.

As part of this project we have used the Microsoft cloud Azure to host the policy server of UCON model. This has enabled us to separate the non-sensitive data from the sensitive data and has helped in achieving resilience by using the replication mechanism to ensure that services are not impacted in the scenarios where a policy server has broken down. Policy Database will also be maintained in cloud and any hospital specific sensitive data has been kept in local network to maintain data security. Cloud load balancing concept has also been used here for effective management of incoming traffic. Load Balancer uses round robin algorithm to route the traffic to respective virtual machines. It provides stickiness within the session which means traffic from a specific session will be sent to the same member of the load balanced set.

1.4 Research Motivation

In today's' ever changing computing environment, there is a need to have a high performance access control model which is resilient and is secure. We have chosen UCON model since in comparison to other traditional access model it provides distinguishing features of continuous evaluation and mutable attributes. Features of cloud have also been explored with respect to scalability, data isolation and pay as you use functionality.

Existing implementation of UCON model do not cater for scenarios where high availability, performance and resilience is required. The objective of this research is to create an enhanced UCON model that can scale effectively as per user demand, perform isolation of sensitive data from non-sensitive data and data transmission between different modules should be via secure model. The model should also be flexible to component failures. E-Healthcare system is one of the most vital areas where such access control system should be implemented since it demands for higher security with respect to patient data, higher availability, capability to cope with node failures etc. Cloud computing has enabled us to achieve these features.

1.5 Goal, Scope and Objective of Research

Goal of research is to enhance the UCON model by adding the following features to it:

1. High Availability

2. Resilience
3. Data Isolation
4. High Performance
5. Data transmission security

Implementation of aforementioned features has been considered as part of this thesis. We have also implemented all the distinguishing features of UCON like obligation handling, re-evaluation of authorization decisions, continuous evaluation, mutability of attributes etc. Separate components have been developed to achieve the desired functionalities.

Developed Access control model has been implemented in e-healthcare domain. Existing implementation of UCON model for this domain is not robust in terms of performance, availability and security. As part of this research, we have developed an enhanced model which caters for all these requirements.

1.6 Thesis Structure

The thesis contains six chapters. Chapter one describes the introduction of different access control models available. This chapter also explains the motivation of Research and focus on important goal, scope and objective of study. Here we have discussed about the distinguishing features of UCON model and improvements required in existing implementations of it.

Chapter 2 describes OASIS XACML along with its usage in existing implementation and structure of existing model with its challenges. This chapter also covers the related research done so far in the field of UCON

Chapter 3 describes the proposed work to overcome the challenges of availability and security in existing implementation. This will cover new architecture proposed which will also include introduction of Microsoft cloud Azure in the new design and all other related components

Chapter 4 contains details of software used for implementing UCON in e-healthcare system. Database schema details along with details of policy server, obligation timer, load balancer and client application. It will also show the final results and comparison of the performance improvement

Chapter 5, describe the conclusion and future work.

Chapter 2
Literature Survey of UCON Model

2.1 OASIS XACML

XACML stands for extensible access control mark-up language, it is an OASIS standard. It is a policy language used to explain general access control requirements and has standard extension points which are used to define new functions, different data types, combining logic etc. The request/response language helps you form a query to check whether given action should be allowed or not and interpret the outcome. The response always includes an answer to the query, it has one of the four values: Permit, Deny, Indeterminate (an error occurred or some required value was missing, hence a decision cannot be made) or Not Applicable (the request can't be answered by this service). PDP (Policy Decision Point) and PEP (Policy Enforcement Point) are two main components of XACML model. PDP is a processing engine that evaluates policies based on the request received. PEP enforces access to a request physically and generates request for policy decision point. PDP and PEP may exist in the same system or may be distributed. An example describing the communication between PDP and PEP-

A subject wants to take an action on an object; it will make a request to suppose a file system or web server which is actually protecting the resource, which we call as policy enforcement point (PEP). Policy enforcement point forms the request using subject's attributes, resource's attribute, action to be performed and other information related to it. Once request is formed, PEP sends it to Policy Decision Point (PDP). PDP analyzes the request and searches for a policy that applies to it, based on the policy applied it returns an answer whether access should be granted or not.

Sun has implemented XACML using a set of JAVA classes that understand XACML language and its rules. Policies are defined using Policy and PolicySet references. A custom class called "finder module" has been used which can be plugged into Policy Decision Point, it helps in retrieving the policies, attribute values etc. Our project implementation adapts this framework for UCON $_{ABC}$ authorization.

XACML advantages over other access control policy languages:

- It is a standard access control policy language that is platform and technology agnostic and can be integrated with any application or language using XML

- Saves time and money for both administrators and developers because there is no need to rewrite their policies in different languages and existing code can be reused.
- XACML allows for easy modifications to support most access control policy use cases and can be extended to cater to new requirements.
- A single XACML policy can provide coverage for multiple resources and thus allows administrators to have a consolidated view of access control policies across the estate.
- XACML supports policy combining algorithms, which allows multiple policy constraints to be applied to create a single, non-conflicting decision based on an input set of parameters.

2.2 Existing Solution Architecture

Below diagram shows the existing architecture for an e-healthcare implementation utilising UCON model. In this model UCON has been implemented using cloud infrastructure. The user platform is on local network, which interacts with the UCON model for making any access control decisions. There are various components of UCON model each with its distinct feature which enables the system to make decisions by applying policies corresponding to the request raised by the subject. The model caters to handling pre and ongoing obligations via the obligation handler, obligations received from Policy decision point are processed here and appropriate commands are issued to interacting modules.

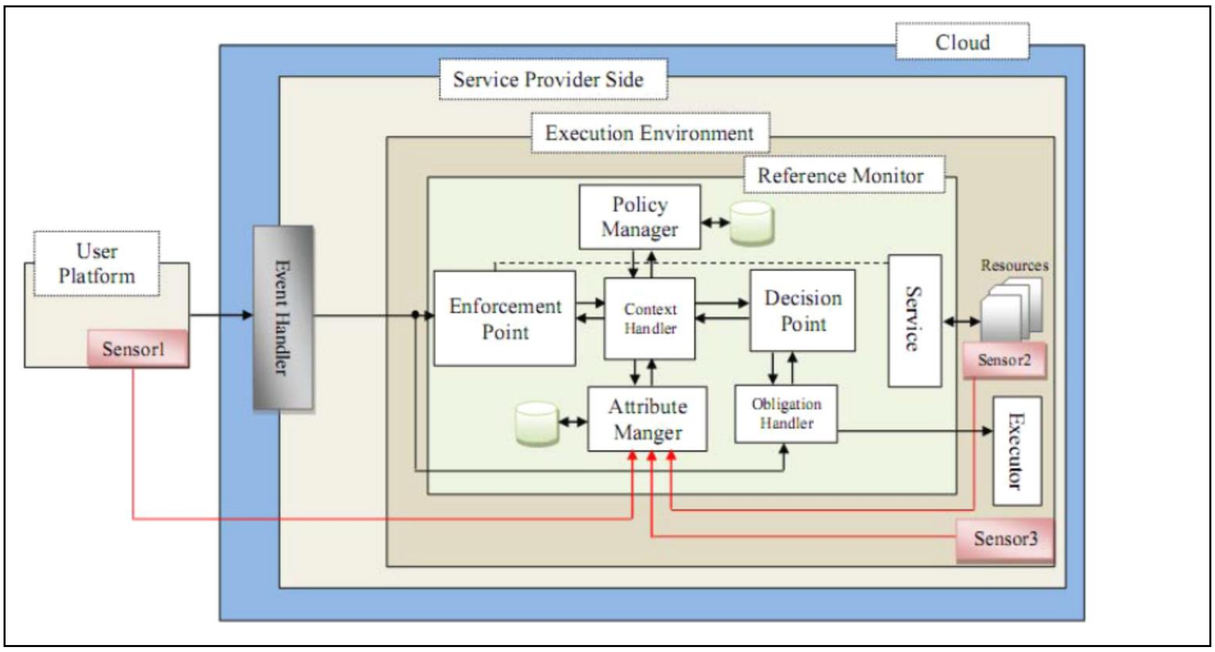


Figure 1: Existing Architecture of UCON Model

Below diagram shows the inner architecture of Obligation Handler

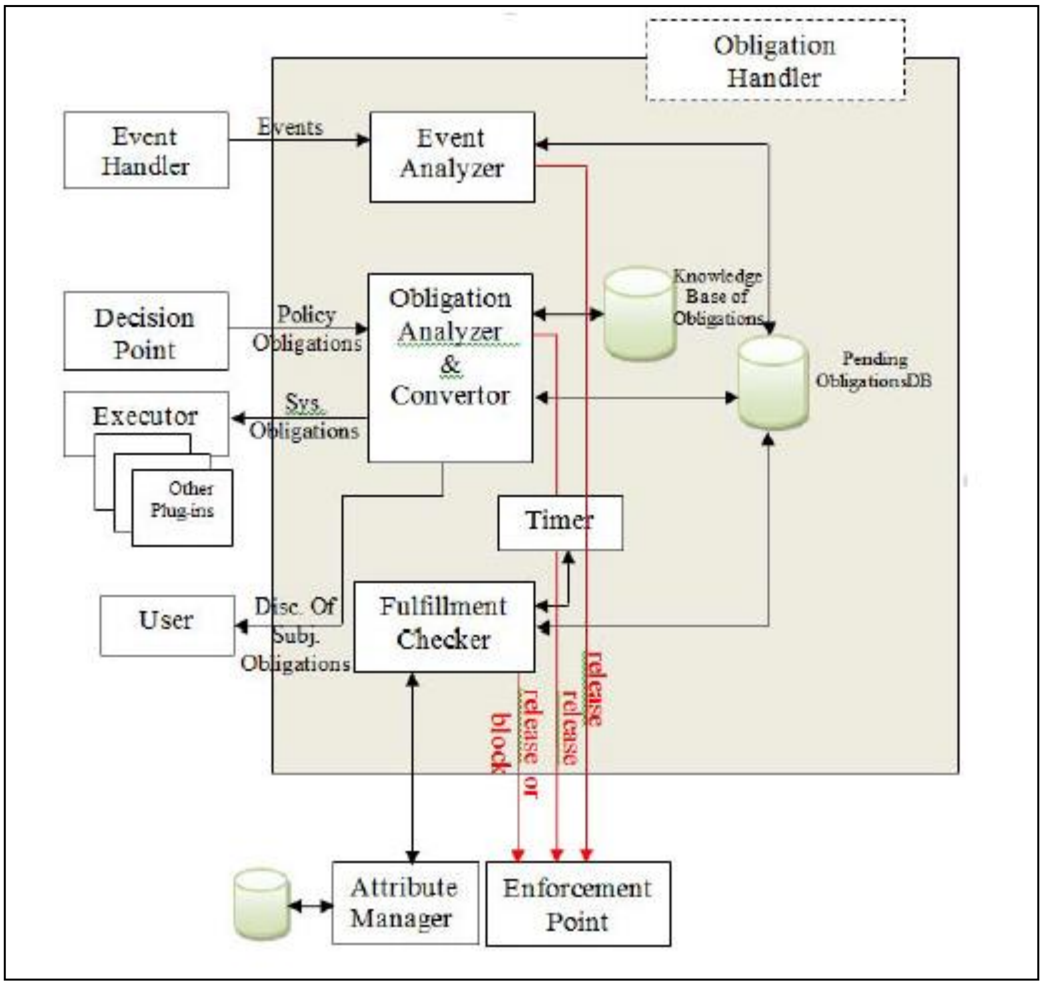


Figure 2: Architecture of Obligation Handler

2.3 Challenges in existing Architecture

Some of the challenges observed in existing architecture are listed below:

1. **Restricted performance due to single policy server:** In the current architecture, a single node is used as a policy server. This reference design does not scale well, as with increase in user and application load, the number of access decisions that require processing would increase exponentially and introduce a higher than desirable waiting time in the process.
2. **System Resilience:** In the event of failure of the policy server or any component within the execution environment, policy decisions can no longer be evaluated, thus creating a system with low resilience to failures. Further, there might be instances where it is desirable to bring one part of the system down for planned maintenance without impacting the overall availability of access control systems.

3. **Security Issues during data transit and at rest:** The existing architecture is implemented in the cloud. Messages from client applications to execution environment and vice-versa containing decision information and potentially confidential resource data traverses organization boundaries when travelling to/from cloud to organization network. This exposes such data to exploitation by maleficent entities.
4. **Data Security Issues:** The current implementation of execution environment does not segregate resources/organizational data from policy data, both being stored in the same cloud environment. In scenarios where tighter control is required on resource data not leaving organizational boundaries, this can pose a challenge.

2.4 Related Research in UCON

Lili Sun and Hua Wang [1] have proposed a usage control model with purpose control to handle private data in e-healthcare system. Purpose is required to access or collect private data in access management system, type of access should change as the purpose changes. Along-with existing components - subject, object, rights, obligations, conditions, authorizations; a new component called purpose was added. Purpose hierarchies were added to the model where upper layer are more general and lower layer are more specific. For eg: “Given Treat” is an upper layer and its lower layer can be “Write Prescription” and “Refer Patients to do X-ray test”. They proposed two versions one for pre-authorization and other for ongoing authorization. For usage control in pre-authorization model following predicate must be true – “if subject s is allowed to access object o with right r based on purpose pu, then the specified condition preA must be true”.

Tina Tanvi, Mehdi Shajari and Peyman Dodangeh [2] have proposed a usage control based architecture for cloud environment. They have proposed a new architecture to perform obligation handling and a new approach for handling attribute mutability using Sun’s OASIS XACML Implementation. The new architecture covers all UCON components and caters to all real world usage control scenarios. This has been done by extending XACML policy language to handle on-going and pre-authorization, re-evaluation of authorization policies, on-going and pre-checks for conditions.

Arlindo Luis Marcon Jr., Altair Olivo Santin, Maicon Stihler, and Juliana Bachtold Jr. [3] have proposed a $UCON_{ABC}$ resilient authorization evaluation for cloud computing. This aims at providing resilience to authorization re-evaluation by dealing with individual exception conditions while maintaining an access control in dynamic cloud environment. Proof of concept of this was done on an e-commerce application. The proposed approach uses contextual information to enable reconfiguration of usage policies and for monitoring of usage SLAs. It ensures when client’s SLA is nearing the contracted amount and additional quota should be added to avoid any SLA breach. It has made the attribute related to authorization of each user more flexible.

Mounira Mshali and Ahmed Serhrouchni [4] have proposed an access control in probative cloud. They have work towards the problem of protecting data from unauthorised access to Cloud. This has been implemented in a government safe project using Hadoop distributed file system. gSafe project deals with externalization of enterprises electronic file system. They have used a new cloud service called Safebox which is used for archiving sensitive documents after a defined period of time. A new actor called Proof Manager has been introduced to manage proof between cloud user and cloud provider. Main objective here is to reduce the risk of malicious access to the sensitive data stored, this has been reduced by encrypting the data using cloud user public key. Encrypted data will also have a security policy in XML form which contains metadata of the stored file and access rights on that file. Feasibility and effectiveness of this model has also been validated.

Dongliango Jiao, Liu Lianzhing, Li Ting and Ma Shilong [5] have presented realization of UCON model based on extended XACML. They have extended PDP and storage mode of policy using extended UCON model (eXUCON). Access request will now not only depend on authorization but also on obligations and conditions. The results of request will now not only return grant or deny but also include the mutable attributes that have changed as a result of the access. They have described the operation of model using components like PAP, PDP, PEP etc; this has also been tested by taking few scenarios.

Fang Pu, Daoquin Sun, Qiying Cao, Haibin Cai and Fan Yang [6] have proposed a pervasive computing context access control based on $UCON_{ABC}$ model. This model not only focuses on authorization but also includes obligations and conditions taking into considerations pre-authentication, pre-condition, pre-obligation, active right and passive right models. They have taken into account contextual information in pervasive computing environment and focused on immutable attributes to produce a CACM model (Context Access Control Model). $CACM_{preA}$ model has pre-authorization that are to be satisfied for usage allowance. $CACM_{preB}$ model has pre-obligations that are to be fulfilled for making a usage decision. $CACM_{preC}$ model has pre-conditions that are to be evaluated before requested rights are given; conditions are associated with environmental conditions that are to be satisfied for usage control. In this paper, this model has been applied in a healthcare scenario

Yonggang Ding and Junhua Zou [7] have presented the application of DRM in $UCON_{ABC.}$ In commercial DRM solutions, usage decisions use user-defined, application level, payment based security policies. This can be realised using UCON model for eg: DRM pay per use using $UCON_{preA}$ payment which is a payment based authorization. In such system user's credit will reduce as per usage and post updates will be required in it. Similar to this there are other scenarios which can be covered via UCON models and hence it proves to have potential for next generation access control.

Tamleek Ali, Mohammad Nauman, Fazl-e-Hadi, Fahad bin Muhaya [8] have described usage control of multimedia content in and through cloud computing paradigm. In this paper, they have presented a new architecture for fine-grained control over usage of protected objects by using the cloud computing environment. They have also worked towards the secure dissemination of multimedia content i,e. if a third party is releasing the data to cloud they need assurance that data is not compromised. Their framework allows resource owner to specify policies related to number of times a resource can be used, time taken per each use etc.

Fengying Wang and Fei Wang [9] have presented the research and application of resource dissemination based on credibility and UCON. It focuses on the distribution of usage rights for digital information. Concept of fuzzy set theories has been used to calculate credibility. This approach gives more control over the level of resource distribution making authorization and revocation process more flexible. A notion called trust degree has been introduced here which evaluates to a number between 0 and 1 based on which grades like complete trust, special trust , general trust and distrust are decided. A trust vector is generated based on five factors and resultant credibility value is calculated. To gain access of a resource subject's credibility should be greater than equal to the credibility of a resource. This is a better approach than certificate based dissemination of usage rights as its provides more flexibility and prevents multi level verifications of certificates.

Zhang Guoping and Gong Wentago [10] have presented the research of access control based on UCON in the Internet of things. Role of UCON in IoT is to provide privacy and authorization flexibility. Through obligations and conditions, decisions will be controlled like usage of resources. IoT has three layered architecture where the top layer is application layer it provides people the application services like logistics monitoring, middle layer is the data transmission network layer such as cloud computing platform or internet and the third layer is data sensing layer consisting of devices like RFID sensors/tags, Smart terminals etc. The Access Control architecture proposed has a trust management centre along with access control model. Trust management centre is used to control the threshold limits defined for devices and services. Access control policies are used to control the access between the device and service. Assessment model based on the concept of IoT is used and many examples have been demonstrated to verify the usage control model.

Jaheong Park and Ravi Sandhu [11] paper on The UCON_{ABC} Usage Control Model. This paper introduces UCON and its concepts. Usage Control is a generalization of access control and covers obligations, conditions, authorizations, mutability and continuity. UCON_{ABC} is considered as next generation access control model which integrates diverse concepts into a unified framework. UCON encompasses traditional access control, trust management and DRM and achieves fine grained access control on digital resources even after their dissemination. This model can be used for both client side and server side control architectures. There are eight core components of UCON_{ABC} model: subject, subject attributes, object, object attributes, conditions, authorizations and obligations. ABCs' are functional predicates that are evaluated for making usage decisions. Paper discusses more on each of the core component.

Chapter 3
Proposed Work

3.1 Problem Statement

Let us consider a scenario wherein policy server architecture implemented on a single node is being stressed by multiple requests per second for authorization decisions. The latency caused by system performance in such instances would have a significant impact downstream in the consuming applications. The absence of multiple worker nodes to share the request load is a limiting factor in system performance.

Furthermore, in the event of failure of any component in the policy server, the entire decision framework suffers downtime resulting in crucial outage.

Considering that the authorization decision system is deployed in the cloud, significant measures have to be taken to ensure that the confidential user data is not compromised in the event of a breach. This is only possible if data isolation is in place with user/resource data being kept at a separate layer. This data is never exposed to the cloud infrastructure.

The main objective of the proposed solution is to address some of the constraints to improve the system in the below mentioned aspects

1. Multi node architecture and Peak Load Performance
2. High availability
3. Data Security and Isolation

3.2 Proposed Solution

As part of this project, we have studied and implemented UCON in the context of e-healthcare domain. Below is the list of entities of the system:

- Subject – Doctor, Visiting Doctor
- Subject's Attributes – Doctor ID, Home Hospital (Hospital ID)
- Object – Patient
- Object's Attribute – Patient's Hospital (Hospital ID)
- Pre-Authorization – Doctors can only see details of Patients of their hospital or they can see details of patient of visiting hospital if obligations are met
- Pre-Obligation - Consent form should be filled by the Visiting doctor in order to access the visiting hospital's patient data

- Ongoing-Obligation - Forms validity is only for 15 days, it needs to be refilled once the tenure ends
- Ongoing-Condition – User's current session validity is constantly re-evaluated once the session is invalid any ongoing and new authorization request from the user are denied
- Pre-Condition – Above condition relating to session validity is also reinforced as a pre-condition which is evaluated by the PDP during access request evaluation.

3.3 New Architecture and its components

The figure below depicts the new architecture proposed for our solution. As shown in the figure, we have redesigned the architecture to split the components into two separate layers.

The core computational modules, which benefit for scaling, are implemented on cloud using Microsoft Azure platform. Enabling modules which help communicate to cloud infrastructure, secure and distribute requests and provide oversight are implemented in the corporate network.

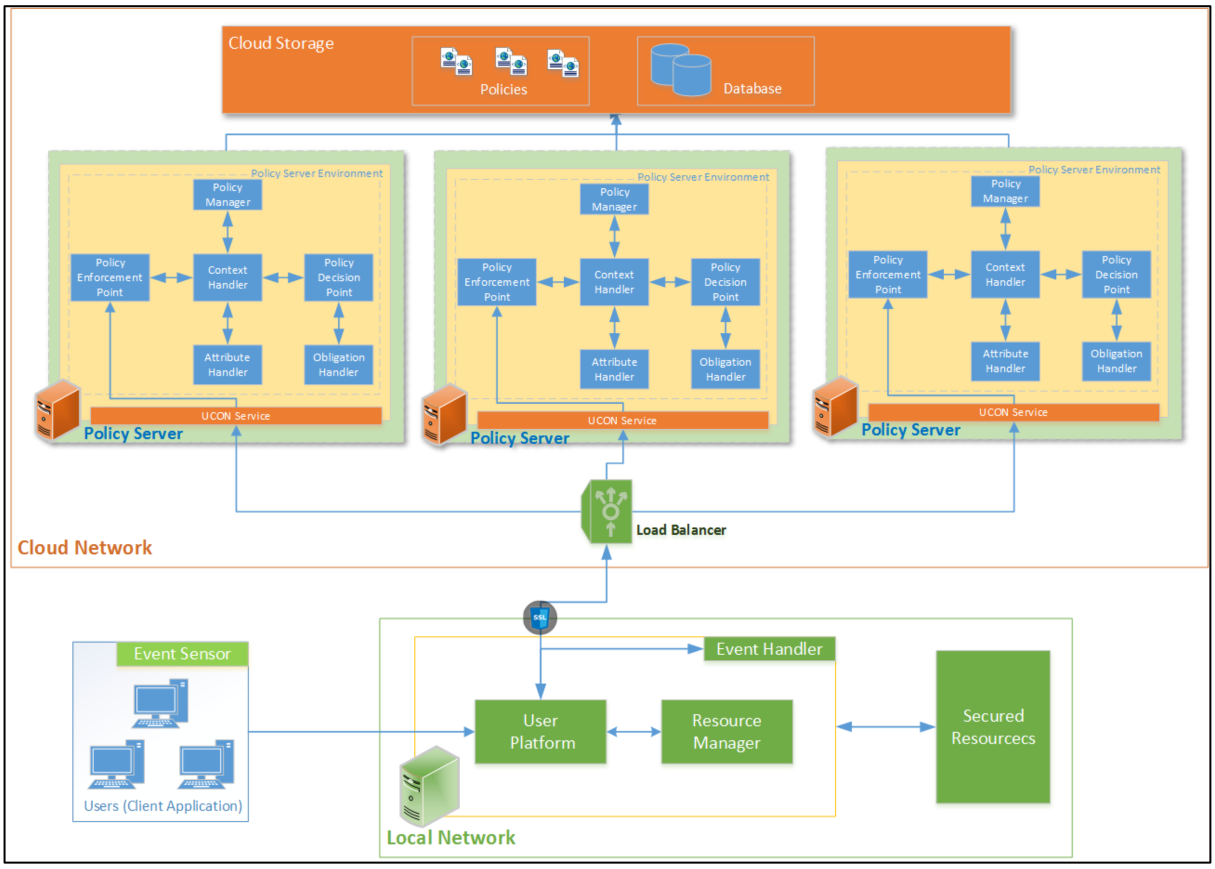


Figure 3: Proposed Architecture of UCON Model

Components parts of the architecture are as follows:

1. **Client application:** Client application is the end user interface which will be consumed by end users of the system. This allows users to query for data/resources present inside the corporate network. How and where the policy decisions that take place to permit access does not affect the functionality of the application and user experience remains unchanged from the original architecture
2. **User Platform:** User platform is the gateway for client application to secure access to the resources requested by the user. User platform is responsible for making access as seamless as possible, while acting as a firewall to prohibit invalid requests.
3. **Resource Manager:** Resource Manager is the module which is invoked by user platform to garner requested resources and in turn provide them back to the client application in a common messaging format.
4. **Secured Resources:** These are the resource for which access needs to be controlled. Secured resources can represent any kind of resources (like files, table data etc.) and can be persisted on file systems, databases, table storage. For our implementation, we have used relational database to persist secured resources.
5. **Load Balancer:** Load balancer is required to distribute incoming access evaluation requests and access evaluation response requests from the user platform to cloud infrastructure. We have implemented load balancing using an Azure Cloud Load Balancer device with sticky session affinity.
6. **Policy Server:** A policy server represents a single, replicable unit that can be scaled up or down as required in the cloud. Policy server internally comprises multiple modules, listed below. Each of these modules in turn has implicit sub modules. Policy server will be described in further details in the following sections, a brief synopsis of each sub-module is as follows
7. **UCON Service:** A load balanced service to distribute access request load across the cloud environment. UCON Service is hosted using IIS Server, installed on each Policy Server.
8. **Policy Execution Environment:** Core Policy evaluation environment implemented by extending SUN XACL framework. This comprises the below sub-components:
   - PEP
   - PDP
   - Obligation Handler
   - Policy Manager
   - Context Handler
   - Attribute Handler

9. Cloud Storage: This is a common locally replicated storage for persisting policies used by the cloud policy server environment. Cloud storage provides for saving both policy data as well as table data used by Policy execution environment.

Chapter 4
Implementation, Testing and Results Analysis

4.1 Software Details of Implemented System

In our implementation, we have used a mix of technologies and platforms as per functional applicability. At a high level, below platforms are used

Server OS: Windows Server 2012

Database: MS SQL Server

Programming Languages: C#.NET, Java

Frameworks: .NET 4.5, SUN XACML, WCF, ASP.NET, XML

Service Hosting: IIS

Cloud Platform: Microsoft Azure

Below table lists specific details for platform against each component:

Component	**Implementation Platform Details**
Client Application	C#.NET
User Platform	C#.NET
Service Proxy	C#.NET
Event Hander	C#.NET
Resource Manager	C#.NET
Secured Resources	MS SQL Server
Load Balancer	Azure Load Balancer
Policy Server	Azure Virtual Machine with Windows 2012 Data Centre Edition
UCON Service	.NET WCF Service
Application Server (Service Host)	IIS
PEP	Java
PDP	Java
Obligation Handler	C#.NET
Policy Manager	Java
Context Handler	Java
Attribute Handler	Java
Policy Storage	Azure File Storage
Table Storage	Azure Database

Table 1: Technology Detail

4.2 Database Schema

Resource Database: The below figure represents a simplistic view of structure required to hold patient case details.

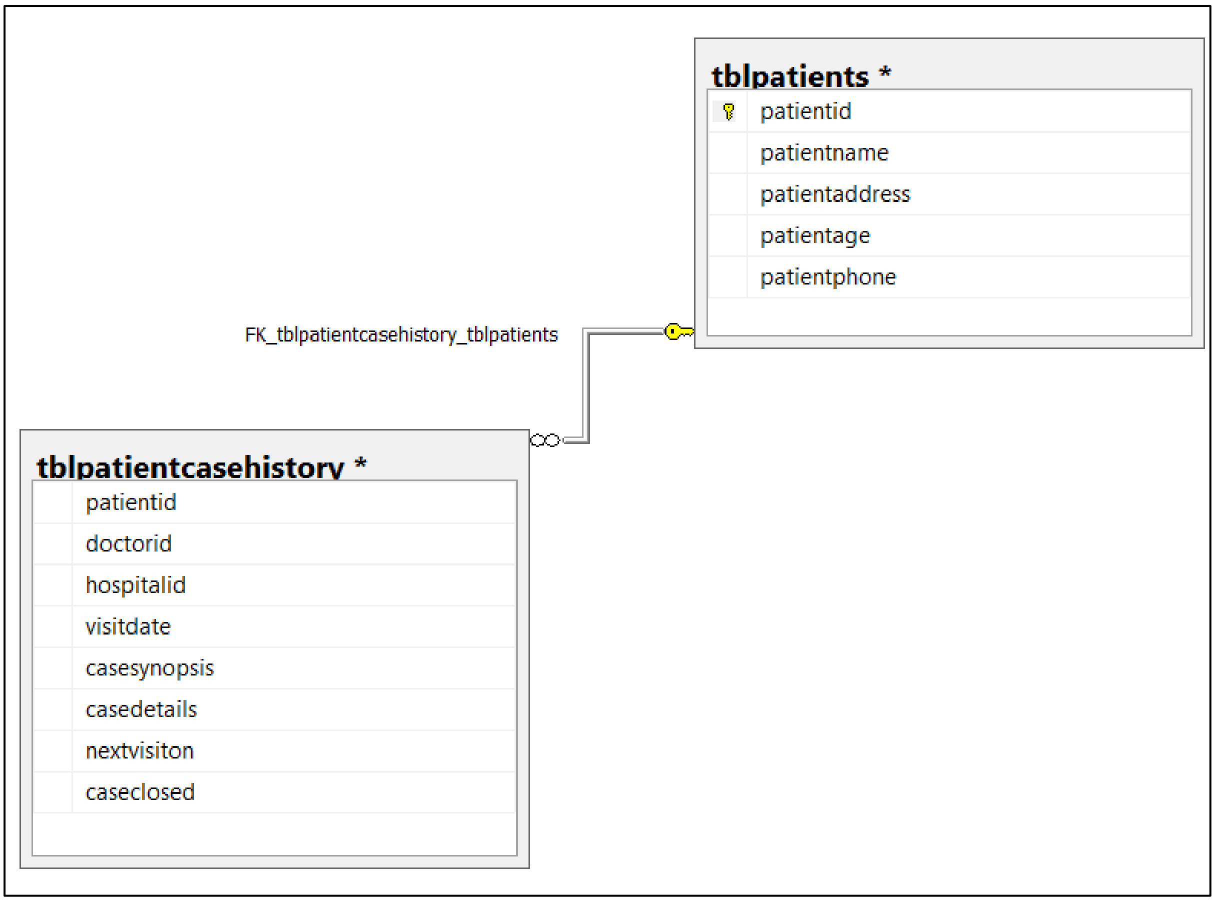


Figure 4: Resource DB Schema

Policy Database: The below figure depicts the database schema for holding policy specific details, augmented by bare minimum details for subjects to help make evaluation decisions. Note that the patient details are not part of this schema, and have been kept as a secured resource under resource database.

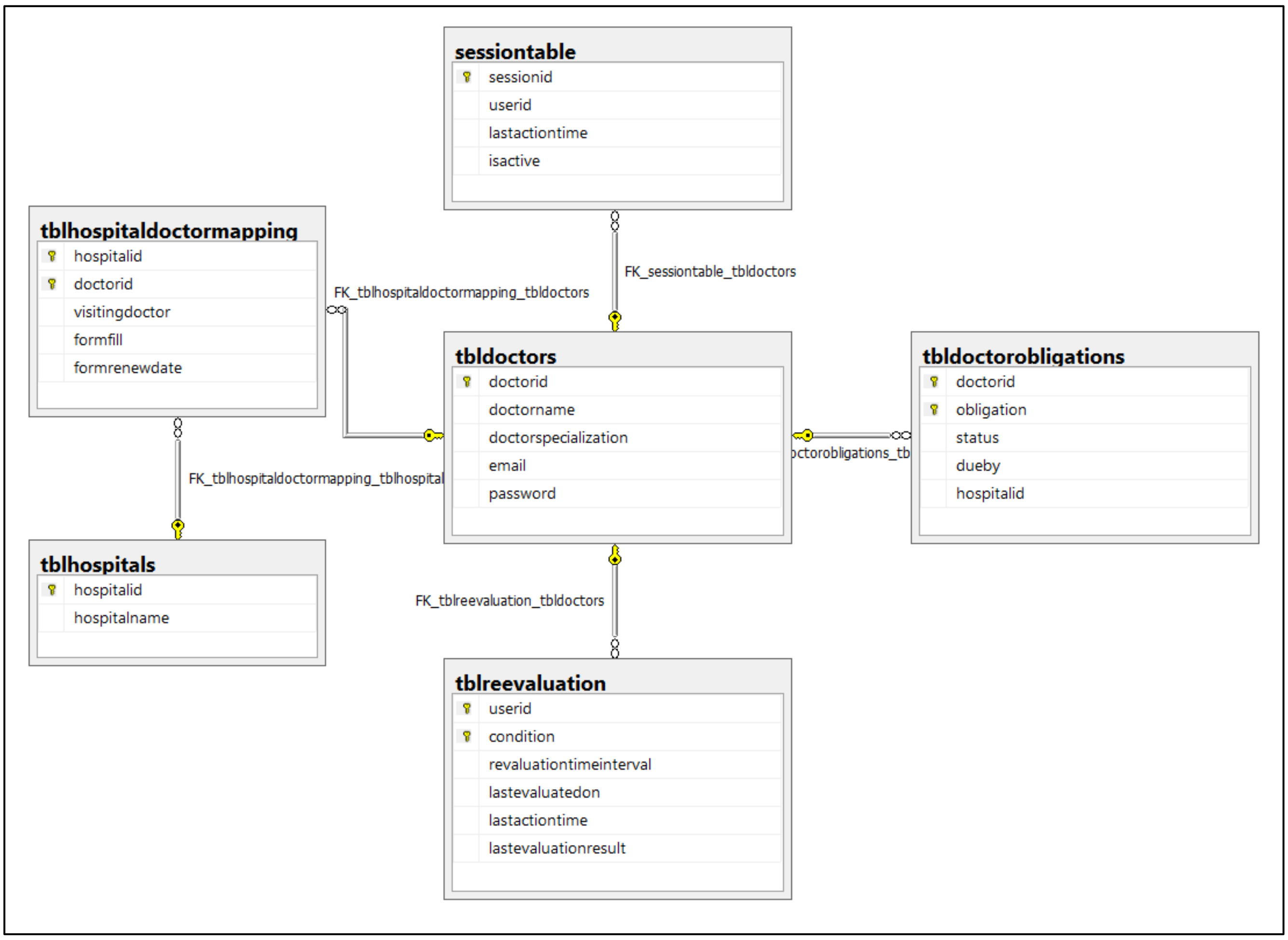


Figure 5: Policy DB Schema

4.3 Messaging Schema

Client to User Platform:

1. Request Schema

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="AccessRequest">

    <xs:complexType>

      <xs:sequence>

        <xs:element name="_action">

          <xs:complexType>

            <xs:sequence>

              <xs:element type="xs:string" name="ActionId"/>

            </xs:sequence>

          </xs:complexType>

        </xs:element>

        <xs:element type="xs:string" name="_requestId"/>

        <xs:element name="_resource">

          <xs:complexType>

            <xs:sequence>

              <xs:element type="xs:string" name="PatientId"/>

              <xs:element type="xs:string" name="ResourceId"/>

            </xs:sequence>

          </xs:complexType>

        </xs:element>

        <xs:element name="_subject">

          <xs:complexType>

            <xs:sequence>

              <xs:element type="xs:string" name="HomeHospital"/>

              <xs:element type="xs:string" name="SubjectId"/>
            </xs:sequence>

          </xs:complexType>

        </xs:element>

      </xs:sequence>

    </xs:complexType>

  </xs:element>

</xs:schema>
```

2. Response Schema

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="ClientResponse">

    <xs:complexType>

      <xs:sequence>

        <xs:element name="Obligations">

          <xs:complexType>

            <xs:sequence>

              <xs:element name="Obligations">

                <xs:complexType>

                  <xs:sequence>

                    <xs:element type="xs:string" name="DoctorId"/>

                    <xs:element type="xs:string" name="Obligation"/>

                    <xs:element type="xs:string" name="Status"/>

                    <xs:element type="xs:string" name="DueBy"/>

                  </xs:sequence>

                </xs:complexType>

              </xs:element>

            </xs:sequence>

          </xs:complexType>
        </xs:element>

        <xs:element type="xs:string" name="AuthorizationResponse"/>

        <xs:element type="xs:string" name="RequestID"/>

        <xs:element name="PatientDetails">

          <xs:complexType>

            <xs:sequence>

              <xs:element type="xs:string" name="PatientName"/>

              <xs:element type="xs:string" name="PatientId"/>

              <xs:element type="xs:string" name="PatientAddress"/>

              <xs:element type="xs:byte" name="PatientAge"/>

              <xs:element type="xs:long" name="PatientPhone"/>

              <xs:element name="CaseHistoryData">

                <xs:complexType>

                  <xs:sequence>

                    <xs:element name="PatientCaseHistory" maxOccurs="unbounded"
minOccurs="0">

                      <xs:complexType>

                        <xs:sequence>

                          <xs:element type="xs:string" name="PatientId"/>

                          <xs:element type="xs:string" name="DoctorId"/>

                          <xs:element type="xs:string" name="HospitalId"/>

                          <xs:element type="xs:dateTime" name="VisitDate"/>

                          <xs:element type="xs:string" name="CaseSynopsis"/>

                          <xs:element type="xs:string" name="CaseDetails"/>

                          <xs:element type="xs:dateTime" name="NextVisitOn"/>

                          <xs:element type="xs:string" name="CaseClosed"/>

                        </xs:sequence>

                      </xs:complexType>

                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

3. PDP Response Schema

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="authorizationDecision">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="PendingObligations">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="doctorObligations">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:string" name="DoctorId"/>
                    <xs:element type="xs:string" name="Obligation"/>
                    <xs:element type="xs:string" name="Status"/>
                    <xs:element type="xs:string" name="DueBy"/>
                  </xs:sequence>
                </xs:complexType>

              </xs:element>

            </xs:sequence>

          </xs:complexType>

        </xs:element>

        <xs:element type="xs:string" name="AuthorizationStatus"/>

        <xs:element type="xs:string" name="SubjectId"/>

      </xs:sequence>

    </xs:complexType>

  </xs:element>

</xs:schema>
```

4. Policy Schema

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="Policy">

    <xs:complexType>

      <xs:sequence>

        <xs:element name="Target">

          <xs:complexType>

            <xs:sequence>

              <xs:element name="Subjects">

                <xs:complexType>

                  <xs:sequence>

                    <xs:element type="xs:string" name="AnySubject"/>

                  </xs:sequence>

                </xs:complexType>
              </xs:element>

              <xs:element name="Resources">

                <xs:complexType>

                  <xs:sequence>

                    <xs:element name="Resource">

                      <xs:complexType>

                        <xs:sequence>

                          <xs:element name="ResourceMatch">

                            <xs:complexType>

                              <xs:sequence>

                                <xs:element name="AttributeValue">

                                  <xs:complexType>

                                    <xs:simpleContent>

                                      <xs:extension base="xs:string">

                                        <xs:attribute type="xs:anyURI"
name="DataType" use="optional"/>

                                      </xs:extension>

                                    </xs:simpleContent>

                                  </xs:complexType>

                                </xs:element>

                                <xs:element name="ResourceAttributeDesignator">

                                  <xs:complexType>

                                    <xs:simpleContent>

                                      <xs:extension base="xs:string">

                                        <xs:attribute type="xs:string"
name="AttributeId" use="optional"/>

                                        <xs:attribute type="xs:anyURI"
name="DataType" use="optional"/>

                                      </xs:extension>

                                    </xs:simpleContent>
                                  </xs:complexType>

                                </xs:element>

                              </xs:sequence>

                              <xs:attribute type="xs:string" name="MatchId"
use="optional"/>

                            </xs:complexType>

                          </xs:element>

                        </xs:sequence>

                      </xs:complexType>

                    </xs:element>

                  </xs:sequence>

                </xs:complexType>

              </xs:element>

              <xs:element name="Actions">

                <xs:complexType>

                  <xs:sequence>

                    <xs:element type="xs:string" name="AnyAction"/>

                  </xs:sequence>

                </xs:complexType>

              </xs:element>

            </xs:sequence>

          </xs:complexType>

        </xs:element>

        <xs:element name="Rule">

          <xs:complexType>

            <xs:sequence>

              <xs:element name="Target">

                <xs:complexType>

                  <xs:sequence>
```

```
                    <xs:element name="Subjects">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:string" name="AnySubject"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="Resources">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element type="xs:string" name="AnyResource"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="Actions">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="Action">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element name="ActionMatch">
                                  <xs:complexType>
                                    <xs:sequence>
                                      <xs:element name="AttributeValue">
                                        <xs:complexType>
                                          <xs:simpleContent>
                                            <xs:extension base="xs:string">
                                              <xs:attribute type="xs:anyURI"
name="DataType" use="optional"/>
```

```
                                            </xs:extension>

                                          </xs:simpleContent>

                                        </xs:complexType>

                                      </xs:element>

                                      <xs:element name="ActionAttributeDesignator">

                                        <xs:complexType>

                                          <xs:simpleContent>

                                            <xs:extension base="xs:string">

                                              <xs:attribute type="xs:string"
name="AttributeId" use="optional"/>

                                              <xs:attribute type="xs:anyURI"
name="DataType" use="optional"/>

                                            </xs:extension>

                                          </xs:simpleContent>

                                        </xs:complexType>

                                      </xs:element>

                                    </xs:sequence>

                                    <xs:attribute type="xs:string" name="MatchId"
use="optional"/>

                                  </xs:complexType>

                                </xs:element>

                              </xs:sequence>

                            </xs:complexType>

                          </xs:element>

                        </xs:sequence>

                      </xs:complexType>

                    </xs:element>

                  </xs:sequence>

                </xs:complexType>

              </xs:element>
              <xs:element name="Condition">

                <xs:complexType>

                  <xs:sequence>

                    <xs:element name="Apply" maxOccurs="unbounded" minOccurs="0">

                      <xs:complexType>

                        <xs:sequence>

                          <xs:element name="Apply">

                            <xs:complexType>

                              <xs:sequence>

                                <xs:element name="SubjectAttributeDesignator">

                                  <xs:complexType>

                                    <xs:simpleContent>

                                      <xs:extension base="xs:string">

                                        <xs:attribute type="xs:anyURI"
name="DataType" use="optional"/>

                                        <xs:attribute type="xs:string"
name="AttributeId" use="optional"/>

                                      </xs:extension>

                                    </xs:simpleContent>

                                  </xs:complexType>

                                </xs:element>

                              </xs:sequence>

                              <xs:attribute type="xs:string" name="FunctionId"
use="optional"/>

                            </xs:complexType>

                          </xs:element>

                          <xs:element name="AttributeValue">

                            <xs:complexType>

                              <xs:simpleContent>

                                <xs:extension base="xs:string">
```

```
                                  <xs:attribute type="xs:anyURI" name="DataType"
use="optional"/>
                                </xs:extension>
                              </xs:simpleContent>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                        <xs:attribute type="xs:string" name="FunctionId"
use="optional"/>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                  <xs:attribute type="xs:string" name="AuthorizationType"
use="optional"/>
                  <xs:attribute type="xs:string" name="FunctionId" use="optional"/>
                  <xs:attribute type="xs:byte" name="ReEvaluationTimePeriod"
use="optional"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute type="xs:string" name="Effect" use="optional"/>
            <xs:attribute type="xs:string" name="RuleId" use="optional"/>
          </xs:complexType>
        </xs:element>
        <xs:element name="Obligations" minOccurs="0">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Obligation" maxOccurs="unbounded" minOccurs="0">
                <xs:complexType>
                  <xs:simpleContent>
                    <xs:extension base="xs:string">
```

```
                      <xs:attribute type="xs:string" name="ObligationId"
use="optional"/>

                      <xs:attribute type="xs:string" name="FulfillOn"
use="optional"/>

                      <xs:attribute type="xs:string" name="PolicyType"
use="optional"/>

                      <xs:attribute type="xs:string" name="ObligationType"
use="optional"/>

                    </xs:extension>

                  </xs:simpleContent>

                </xs:complexType>

              </xs:element>

            </xs:sequence>

          </xs:complexType>

        </xs:element>

      </xs:sequence>

      <xs:attribute type="xs:string" name="AuthorizationType" use="optional"/>

      <xs:attribute type="xs:string" name="PolicyId" use="optional"/>

      <xs:attribute type="xs:string" name="ReEvaluationProcess" use="optional"/>

      <xs:attribute type="xs:byte" name="ReEvaluationTimePeriod" use="optional"/>

      <xs:attribute type="xs:string" name="RuleCombiningAlgId" use="optional"/>

      <xs:attribute type="xs:byte" name="Version" use="optional"/>

    </xs:complexType>

  </xs:element>

</xs:schema>
```

4.4 User Platform

User platform is the gateway for client application to secure access to the resources requested by the user. User platform is responsible for making access as seamless as possible, while acting as a firewall to prohibit invalid requests. The client application connects to user platform, which is implemented in the secure internal network. User platform is responsible for following key functions

1. Message routing: User platform listens to request messages from Client applications and forwards requests to Cloud service.
2. Event Handling: User platform is responsible for time bound processing of events that are raised by client sensors or external sensors which impact authorization decisions
3. Cloud Service Invocation: User platform invokes cloud service in a seamless manner, abstracting the multi node complexities from downstream service consumers (client applications)
4. Resource Handling: User platform encapsulates resource handling, for performing CRUD operations on resource data.

User platform encapsulates the below components. Each component is briefly described

Service Proxy:

Service Proxy module is responsible for communication between user platform and Cloud service (UCON Service). Service Proxy abstracts the service connectivity details from user platform and helps create service oriented architecture. Since the connection bindings are configuration driven, they can easily be modified on the fly to redirect to a new instance in the event of planned downtime/maintenance window. Service Proxy connects to UCON Service through a WCF proxy class over HTTPS binding.

Event Handler:

Over the lifetime of user request and user session, various events might happen which impact authorization decisions either directly or indirectly. Any change in state that raises such events needs to be handled in a time bound manner and impacted components be notified of the same such that decisions re-evaluations or amendments take place. Event handler is responsible for implementing such changes based on event notifications by the Client Side Event Sensors or external sensors. When a new event is raised, event handler would take a pre-determined action on the event.

Resource Manager:

Resource Manager is responsible for taking care of resource fetch and save requests. Once a authorization decision is received by User Platform which would allow resource details to be shared with the requesting user, the User Platform invokes resource manager with necessary information to fetch the required resource details. These details are then passed downstream to the consuming application to be presented to the requesting user. Client application by itself has no access to the resource data. Using a modular architecture, the resource manager can be swapped/upgraded in the event of change in storage strategy.

Secured Resources:

Secured resources represent the end data that is being protected from unauthorized access. This data can be accessed only via resource manager, which is called by user platform to from CRUD (Create, Read, Update, Delete) operations, based on a valid authorization token being received from Policy Execution environment following a resource request made from a client application.

4.5 Policy Server and Obligation Timer

In the architecture diagram depicted in the preceding section, Policy server represents a scalable and pluggable entity. This architecture enables the policy infrastructure to scale out in a cost effective and time bound process. Furthermore, the architecture provides for in-built resilience to node failures, zero downtime for any planned/unplanned maintenance and improved peak load performance.

The below sections describe each component of the Policy server in some detail:

UCON Service:

UCON Service is a load balanced Windows Communication Foundation (WCF) service that creates a messaging channel between PEP and User Platform. This service is load balanced to distribute access request load across the cloud environment. UCON Service is hosted using IIS Server, installed on each Policy Server. UCON Service utilises SSL over HTTP to create a secure channel for communications between Cloud infrastructure and User platform.

Policy Execution Environment:

Policy execution environment is the set of modules which are responsible for evaluation of incoming access requests against stipulated policies and communicating the end response back to requester. In our application architecture, the requests come from user platform; however the same architecture can serve any other application which requires capability for access control to be implemented. Policy execution environment comprises these sub-modules:

1. Policy Enforcement Point (PEP)

2. Policy Decision Point (PDP)
3. Context Handler
4. Policy Manager
5. Attribute Handler
6. Obligation Handler

Functionality for each module is described below

Policy Enforcement Point (PEP)

The Policy Enforcement point is the entry point to access the policy evaluation capabilities. PEP intercepts any evaluation requests and issues decision requests to the Policy Decision Point. Furthermore, PEP is responsible for applying any policy specific customizations to the decisions evaluated and sent by PDP.

Policy Decision Point (PDP)

The Policy Decision Point is responsible for evaluating access requests and deliver authorization decisions. As depicted in the application architecture, Policy Decision Point evaluates incoming access requests receive from the Policy Enforcement Point against the Access Control Policies that are applicable for the specific request. These policies include constraints for Conditions and obligations which can impact the authorization decision. During the policy evaluation, the Policy Decision point communicates with Attribute Manager as required to gain access to attributes relating to Subject, Object and Environment properties. These attributes may be part of the incoming request (like user id, hospital id etc.) or may be environment/system attributes (like System Date Time etc.). Once PDP has sufficient data to evaluate the request, it creates a response and sends the response to PEP for further action.

Context Handler

Context Handler is the component in SUN Implementation of OASIS proposed XACML architecture. This component handles inter module communication and provides a duplex conversion capability for the messages being passed. Context handler is implicitly invoked by Policy Enforcement Point, Policy Decision Point and other modules when passing messages during access request evaluation lifecycle. The messages passed are type safe objects which are cast implicitly by Context Handler.

Policy Manager

Policy Manager is responsible for discovering, loading and unloading policies in the execution environment. This is several pluggable sub-components that provide for policy storage utilising disparate storage mechanisms. These components can discover applicable policies during run time based on configuration and provide them to the Policy execution environment for loading. Policy Manager also determines the attribute set which is present in policies and is responsible for communicating this to Policy attribute manager; so that the required attribute infrastructure can be initialised to serve attribute query requests.

Attribute Handler

Attribute handler is responsible for creating an infrastructure that can service requests for attribute information relating to subject, resource and environment. Attribute handler provides an extensible framework that can be amended to involve additional attribute query sources as per requirements. Attribute handler receives updates to information about these attributes from event sensors and is responsible for persisting these changes to storage. Attribute handler is also responsible for ensuring that these changes are communicated to the other modules which require these in evaluating decisions. For sensitive attributes, Attribute handler must inform Decision Point in the event that there is a change is attribute value. For low sensitivity attributes, Decision Point will directly request Attribute handler for updated attribute value as re-evaluation takes place.

Obligation Handler

This component receives obligations from Decision Point and processes them, then issue appropriate commands to the related modules. Obligation handler consists of several subsystems which are described as follows

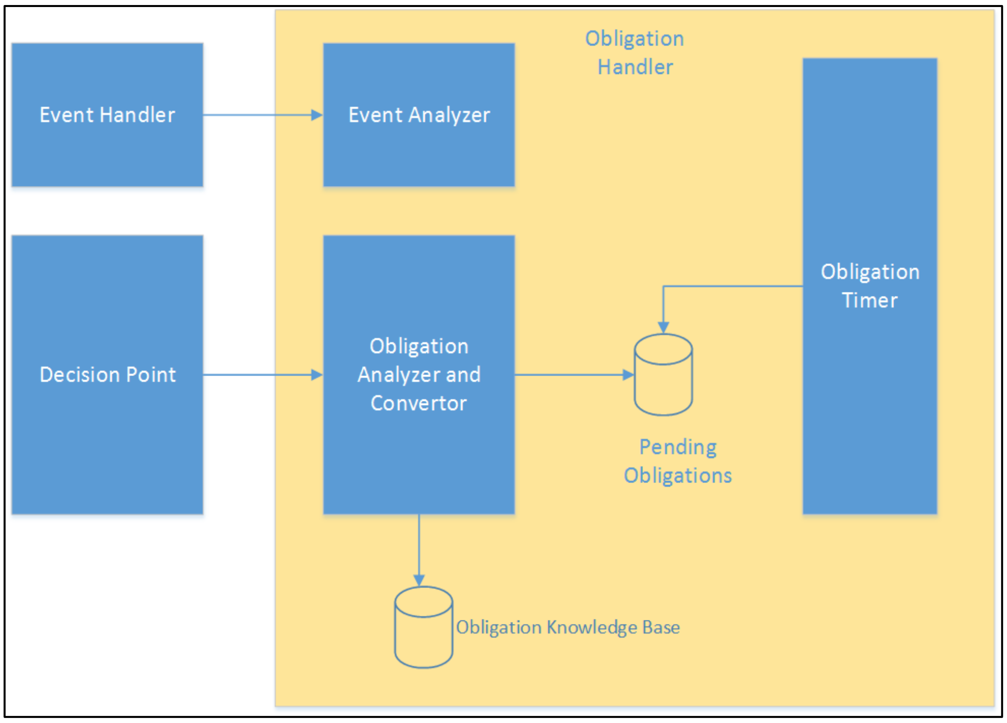


Figure 6: Components of Obligation Handler

A. Event Handler: This is an external component which is part of user platform that communicates with obligation handler whenever an event occurs. These events can cause an obligation to be fulfilled or reopen.

B. Event Analyser: Event Analyser checks whether there is an obligation in the pending obligations database to be fulfilled because of this event or not. If occurrence of one event implies that one or more obligations will be fulfilled, the Event Analyser deletes the obligation from the pending obligations database.

C. Obligation Analyser and Convertor: After analysing the received obligations from the Decision Point, this module sends a description of subject obligations to the user and also system obligations to the system. Before PEP grants privilege to the user, it must intercept results from Obligation Handler if there is an obligation in it. After analysing obligations, if there were no pending obligations with zero deadlines, the grant command will be sent to the user.

D. Obligation Timer: Obligation timer is responsible for obligation maintenance in the pending obligation database.

Cloud Storage:

Cloud storage consists of two main components hosted on Azure to save data. Both these components are configured to provide implicit redundancy, so that in the event of a failure at any level, decision making is not impacted.

1. Policy Storage: Policy storage uses Azure File Storage to save and read policy definition files. These files are created by policy administrators and are secured from external access. Only Policy servers have access to these files.
2. Table Storage: Relational data required in policy evaluation process is stored in Azure DB.

4.6 Load Sharing

Our architecture utilizes multiple Policy Servers to distribute load of policy evaluation requests and a messaging service for enabling communication between policy servers and user platform. To enable request routing and load sharing, we have implemented load balancing using an Azure Load Balancer.

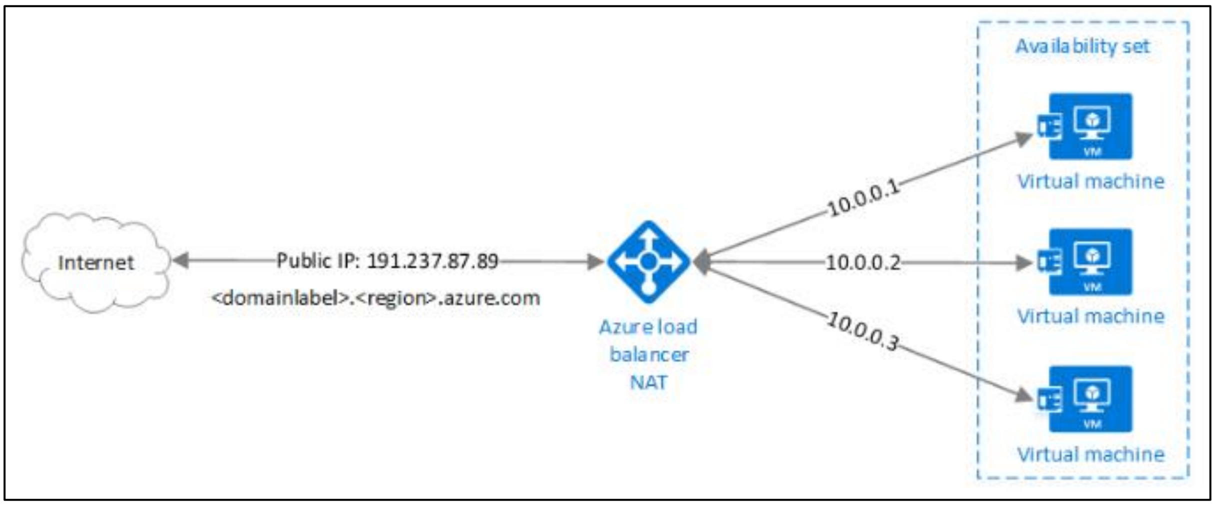


Figure 7: Microsoft Azure Load Balancer

Session Types: We have configured Azure Load Balancer to use a 2 tuple (Source IP, Destination IP) routing configuration for traffic routing to all policy server nodes. By using Source IP affinity, this ensures that any requests initiated from one client are directed to the same policy server node throughout the session lifecycle unless the node is down.

Availability set: Each policy server in the cloud infrastructure is part of the same availability set. This ensures that at all given instances, at least one policy server is up to serve requests. Note that such downtime for remaining servers is only a by-product of any maintenance activity and not a usual scenario. Availability set negates any overall downtime by keeping at least one server in the set available throughout the downtime.

Backend Pool: Each Policy server in the availability set is part of a backend pool. The backend pool is the set of machines that are load balanced by the azure load balancer.

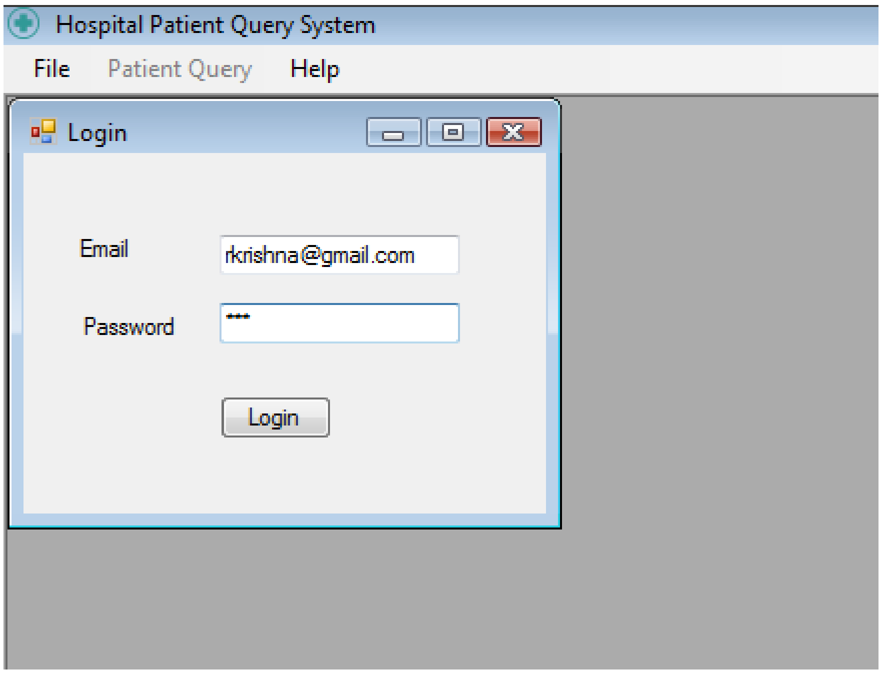
Http Probes and Heartbeat Monitoring: the load balancer constantly monitors each policy server note and in the event of a server being non responsive or not available, stops routing requests to that node.

4.7 Client Application

Client Interface comprise of a client used by end users to perform data queries and various event sensors that relate event information to user platform for processing. We have created a sample client application to simulate patient data queries performed by doctors on the secured resource data. We have also created a sample testbed to measure performance of the scalable policy server architecture.

4.8 Glance at the Working Model

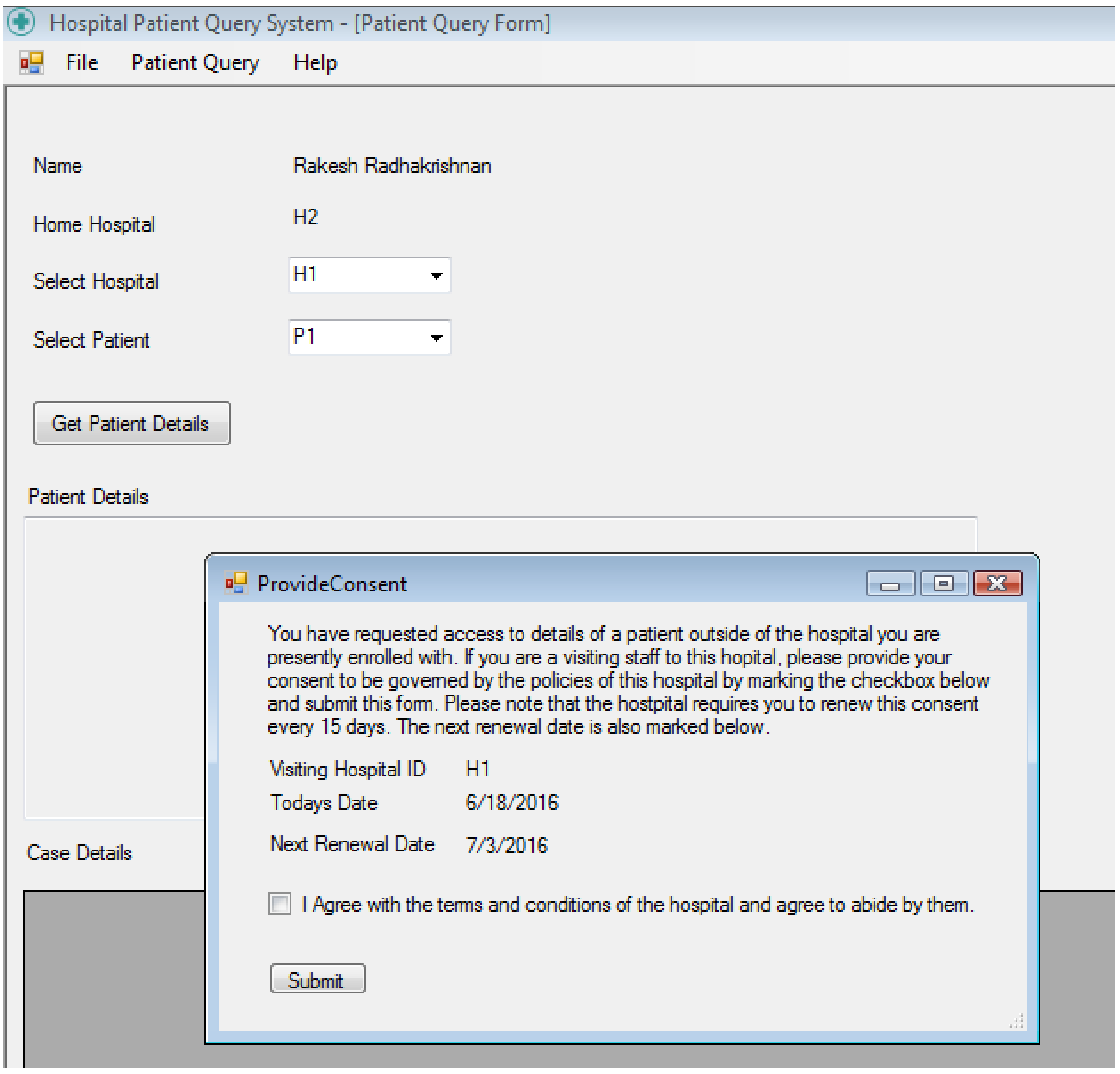
Client Application login Page – Page used by doctor for logging into the client application



Email: rkrishna@gmail.com
Password: ***

Figure 8: Login Page

Hospital Patient Query System – This page is used to fetch patient data. In case of pending obligation, a prompt like the one mentioned below will appear



Name Rakesh Radhakrishnan

Home Hospital H2

Select Hospital H1

Select Patient P1

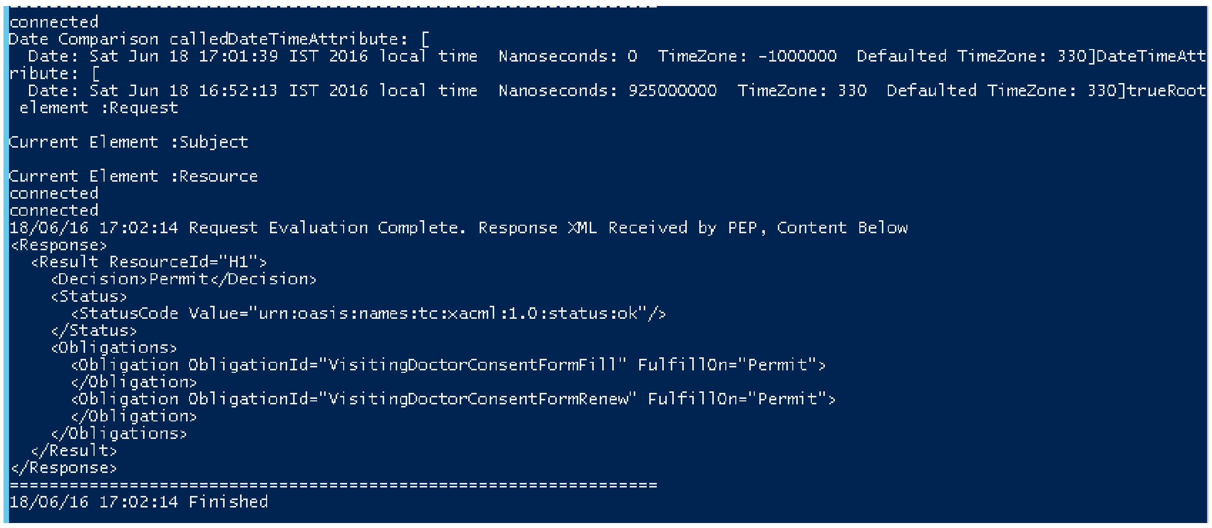
You have requested access to details of a patient outside of the hospital you are presently enrolled with. If you are a visiting staff to this hopital, please provide your consent to be governed by the policies of this hospital by marking the checkbox below and submit this form. Please note that the hostpital requires you to renew this consent every 15 days. The next renewal date is also marked below.

Visiting Hospital ID	H1
Todays Date	6/18/2016
Next Renewal Date	7/3/2016

☐ I Agree with the terms and conditions of the hospital and agree to abide by them.

Figure 9: Hospital Patient Query System

Policy Server – Below Screenshot shows the request processing by UCON Policy Server.



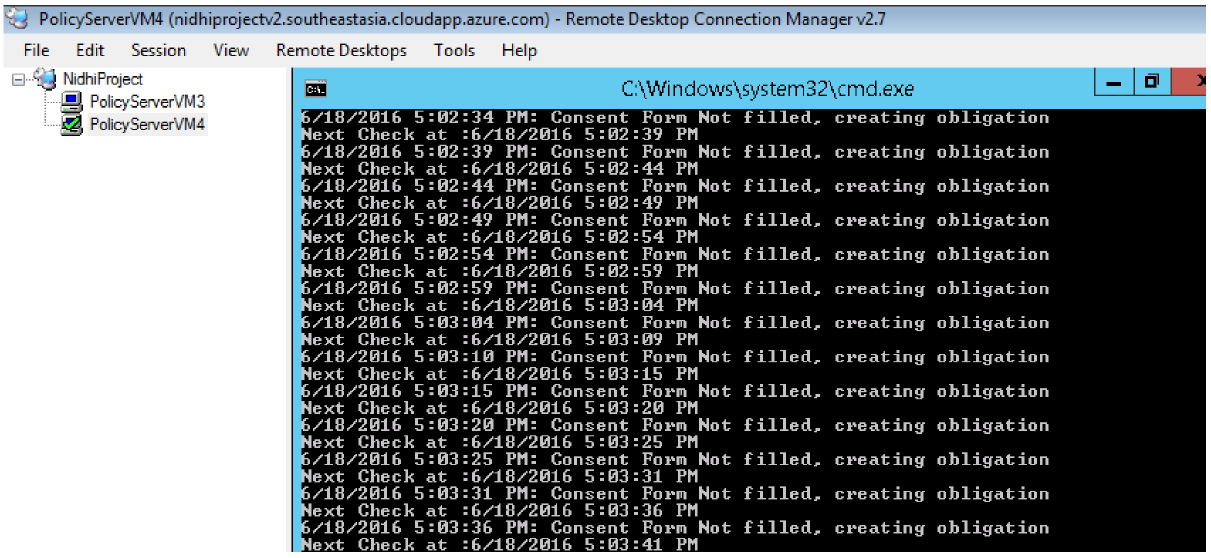
```
------------------------------------------------------------------
connected
Date Comparison calledDateTimeAttribute: [
  Date: Sat Jun 18 17:01:39 IST 2016 local time  Nanoseconds: 0  TimeZone: -1000000  Defaulted TimeZone: 330]DateTimeAtt
ribute: [
  Date: Sat Jun 18 16:52:13 IST 2016 local time  Nanoseconds: 925000000  TimeZone: 330  Defaulted TimeZone: 330]trueRoot
 element :Request

Current Element :Subject

Current Element :Resource
connected
connected
18/06/16 17:02:14 Request Evaluation Complete. Response XML Received by PEP, Content Below
<Response>
  <Result ResourceId="H1">
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
    <Obligations>
      <Obligation ObligationId="VisitingDoctorConsentFormFill" FulfillOn="Permit">
      </Obligation>
      <Obligation ObligationId="VisitingDoctorConsentFormRenew" FulfillOn="Permit">
      </Obligation>
    </Obligations>
  </Result>
</Response>
==================================================================
18/06/16 17:02:14 Finished
```

Figure 10: UCON Request Processing

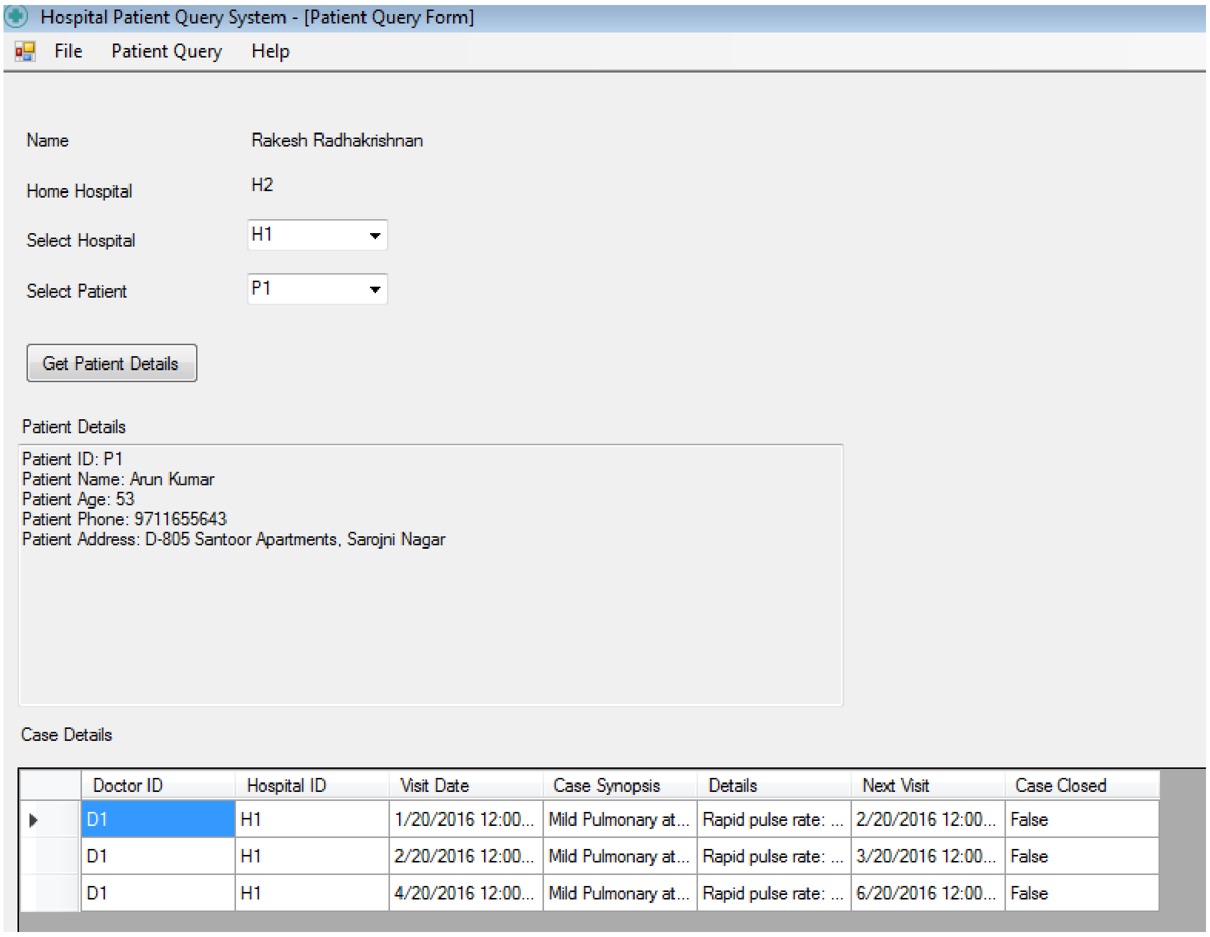
Obligation Handler – Below screenshot shows the processing of pending and completed obligation



PolicyServerVM4 (nidhiprojectv2.southeastasia.cloudapp.azure.com) - Remote Desktop Connection Manager v2.7
File
Edit
Session
View
Remote Desktops
Tools
Help
NidhiProject
PolicyServerVM3
PolicyServerVM4
C:\Windows\system32\cmd.exe

```
6/18/2016 5:02:34 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:02:39 PM
6/18/2016 5:02:39 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:02:44 PM
6/18/2016 5:02:44 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:02:49 PM
6/18/2016 5:02:49 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:02:54 PM
6/18/2016 5:02:54 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:02:59 PM
6/18/2016 5:02:59 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:03:04 PM
6/18/2016 5:03:04 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:03:09 PM
6/18/2016 5:03:10 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:03:15 PM
6/18/2016 5:03:15 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:03:20 PM
6/18/2016 5:03:20 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:03:25 PM
6/18/2016 5:03:25 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:03:31 PM
6/18/2016 5:03:31 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:03:36 PM
6/18/2016 5:03:36 PM: Consent Form Not filled, creating obligation
Next Check at :6/18/2016 5:03:41 PM
```

Figure 11: Obligation Handler



Name	Rakesh Radhakrishnan
Home Hospital	H2
Select Hospital	H1
Select Patient	P1

Patient Details

Patient ID: P1
Patient Name: Arun Kumar
Patient Age: 53
Patient Phone: 9711655643
Patient Address: D-805 Santoor Apartments, Sarojni Nagar

Case Details

Doctor ID	Hospital ID	Visit Date	Case Synopsis	Details	Next Visit	Case Closed
D1	H1	1/20/2016 12:00...	Mild Pulmonary at...	Rapid pulse rate: ...	2/20/2016 12:00...	False
D1	H1	2/20/2016 12:00...	Mild Pulmonary at...	Rapid pulse rate: ...	3/20/2016 12:00...	False
D1	H1	4/20/2016 12:00...	Mild Pulmonary at...	Rapid pulse rate: ...	6/20/2016 12:00...	False

Figure 12: Result of Patient Query Request Processing

4.9 Final Result

In this paper, we have extended the existing implementation of UCON architecture to create a significantly scalable and resilient Augmented UCON platform. The augmented UCON architecture caters to our core requirements of access control, pre and ongoing authorizations and obligations, while augmenting the model with capabilities to overcome challenges identified in Section: Challenges in existing Architecture, re-iterated below.

1. **Restricted performance due to single policy server:** In the current architecture, a single node is used as a policy server. This reference design does not scale well, as with increase in user and application load, the number of access decisions that require processing would increase exponentially and introduce a higher than desirable waiting time in the process.
2. **System Resilience:** In the event of failure of the policy server or any component within the execution environment, policy decisions can no longer be evaluated, thus creating a system with low resilience to failures. Further, there might be instances where it is desirable to bring

one part of the system down for planned maintenance without impacting the overall availability of access control systems.

3. **Security Issues during data transit and at rest:** The existing architecture is implemented in the cloud. Messages from client applications to execution environment and vice-versa containing decision information and potentially confidential resource data traverses organization boundaries when travelling to/from cloud to organization network. This exposes such data to exploitation by maleficent entities.
4. **Data Security Issues:** The current implementation of execution environment does not segregate resources/organizational data from policy data, both being stored in the same cloud environment. In scenarios where tighter control is required on resource data not leaving organizational boundaries, this can pose a challenge.

We have tested our multi node augmented UCON architecture and charted a demonstrable improvement in system efficiency and accuracy. The system was tested in both conventional (single node) and load balanced multi node configurations. We will describe the results in the below sections.

As part of test run, we created a test bed to accurately load the system and determine performance. The test bed is driven by the following parameters from the system:

1. Batch Start size: Each parallel run starts with a minimum batch size and ramps up to a maximum batch size. The batch size determines the count of parallel requests being invoked.
2. Batch Step Size: Once all parallel requests are serviced, batch size is increased in a pre-determined increment determined by the step size
3. Batch Maximum Size: For a test run, maximum size caps the batch size increments to a ceiling level.

The following values were chosen for the above parameters during test run

Test ID	Batch Start Size	Batch Step Size	Batch Max Size
1	10	10	100
2	100	100	1000
3	500	500	5000

Table 2: Test Values Taken

For each run, the test bed captures and persists the below information

1. Start Time: Time when the request was sent to Policy Execution environment by the test bed
2. End Time: Time when a response was received by the test bed for request
3. Number of parallel requests: Number of parallel requests being sent to Policy environment
4. Average Response Time: Average response time across all parallel requests in the batch

5. Request Status (Serviced/Delayed): Response status for the request. In instance where Policy execution environment is operating at its threshold capacity, servicing of request is delayed pending resource availability

Single Node Results

The below tabular data demonstrates the performance of the UCON model in a single node configuration. These figures will serve as a baseline to measure improvements in the following sections

Count	Success	Failure	Total Time	Average Time
20	20	0	20984	1049.2
30	30	0	34170	1139
40	40	0	51989	1299.725
50	50	0	67913	1358.26
60	60	0	91494	1524.9
70	70	0	110693	1581.329
80	80	0	138969	1737.113
90	90	0	193054	2145.044
300	300	0	754493	2514.977
400	400	0	1071216	2678.04
500	500	0	2330865	4661.73
500	500	0	2143435	4286.87
600	600	0	2584060	4306.767
700	700	0	4193378	5990.54
800	799	1	10730294	13412.87
900	823	5	30921998	34357.78
1500	1499	7	18125425	12083.62
2000	1968	32	37706156	18853.08
2500	2449	51	54024498	21609.8
3000	2929	69	83665085	27888.36
3500	3466	71	80148064	22899.45
4000	3927	73	114277956	28569.49
4500	4442	80	110713780	24603.06

Table 3: Single Node Test Results

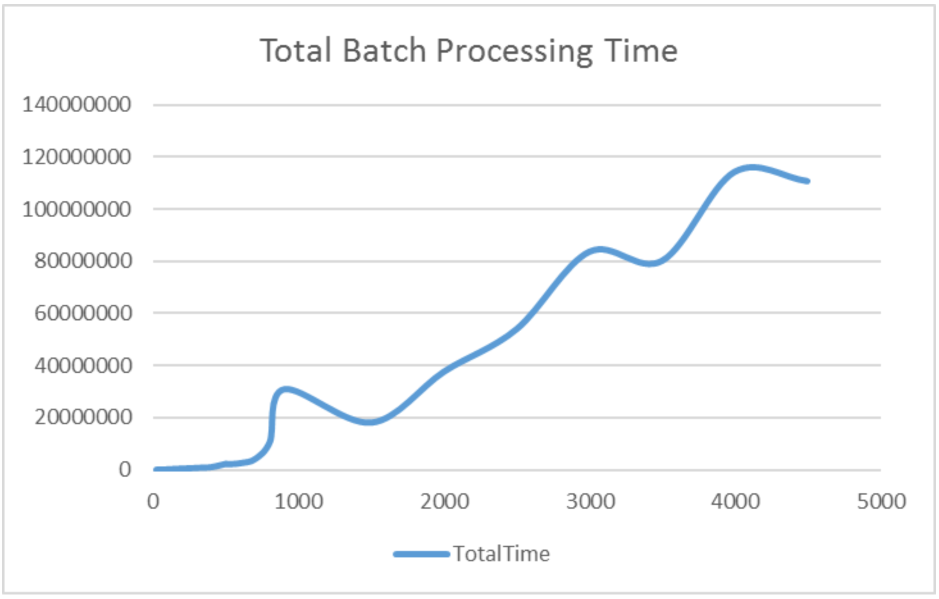


Figure 13: Single Node Batch Processing Time

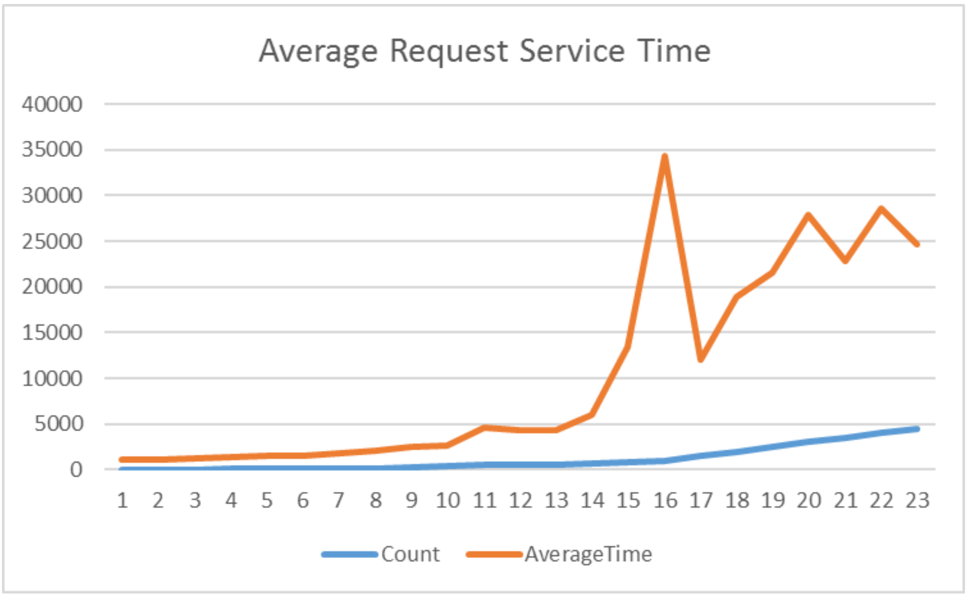


Figure 14: Single Node Avg Request Service Time

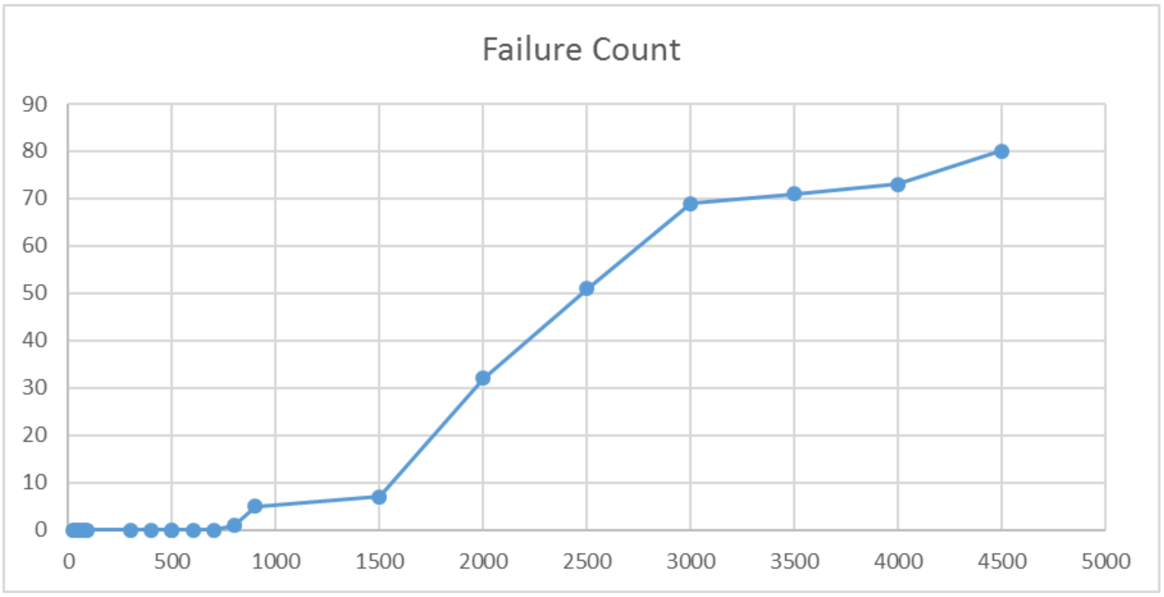


Figure 15: Single Node Failure Count

Multi Node Results

The below tabular data demonstrates the performance of the UCON model in a multi-node configuration.

Count	Success	Failure	Total Time	Average Time
10	10	0	7968	1593.6
20	20	0	8222	822.2
30	30	0	16227	1081.8
40	40	0	20261	1013.1
50	50	0	27013	1080.5
60	60	0	35576	1185.9
70	70	0	40681	1162.3
80	80	0	50478	1262.0
90	90	0	55366	1230.4
100	100	0	159971	3199.4
200	200	0	191455	1914.6
300	300	0	305450	2036.3
400	400	0	480862	2404.3
500	496	4	909142	3636.6
600	594	6	1357933	4526.4
700	700	0	1722289	4920.8
800	800	0	2979013	7447.5
900	898	2	3985406	8856.5
1000	1000	0	1501454	3002.9
1500	1500	0	3124257	4165.7
2000	2000	0	5933025	5933.0

2500	2500	0	10996193	8797.0
3000	3000	0	15284630	10189.8
3500	3498	2	24185084	13820.0
4000	3974	26	39666117	19833.1

Table 4: Multi Node Test Results

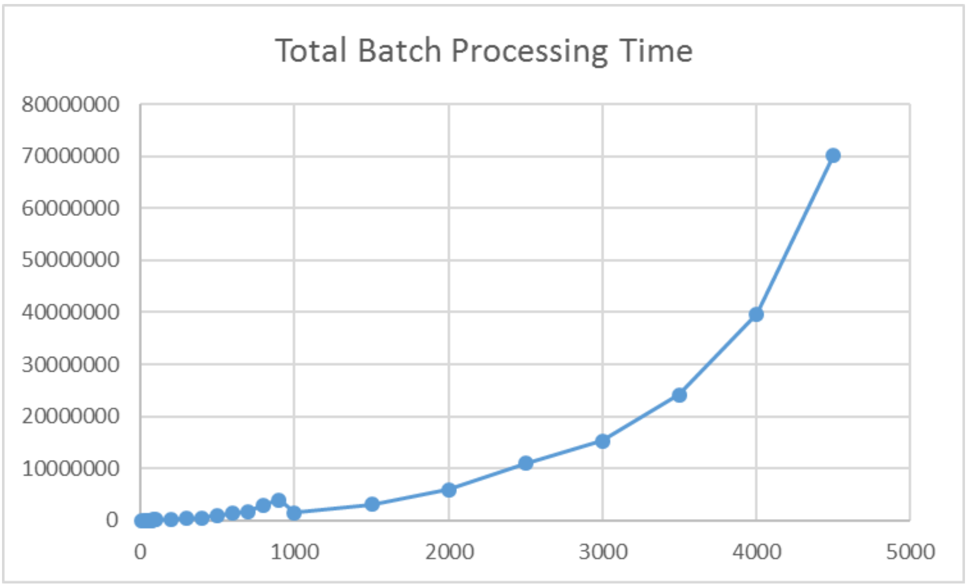


Figure 16: Multi Node Total Batch Processing Time

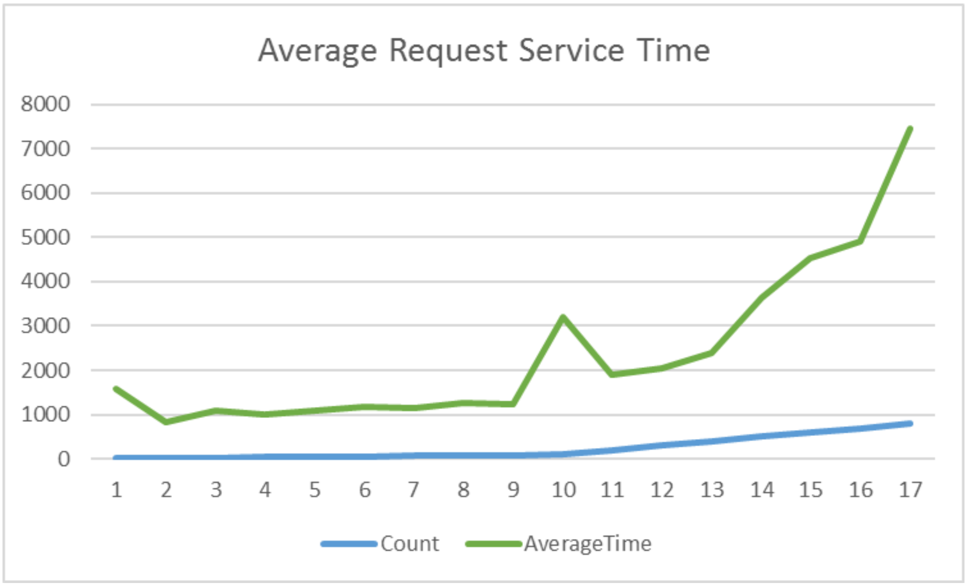


Figure 17: Multi Node Avg Request Service Time

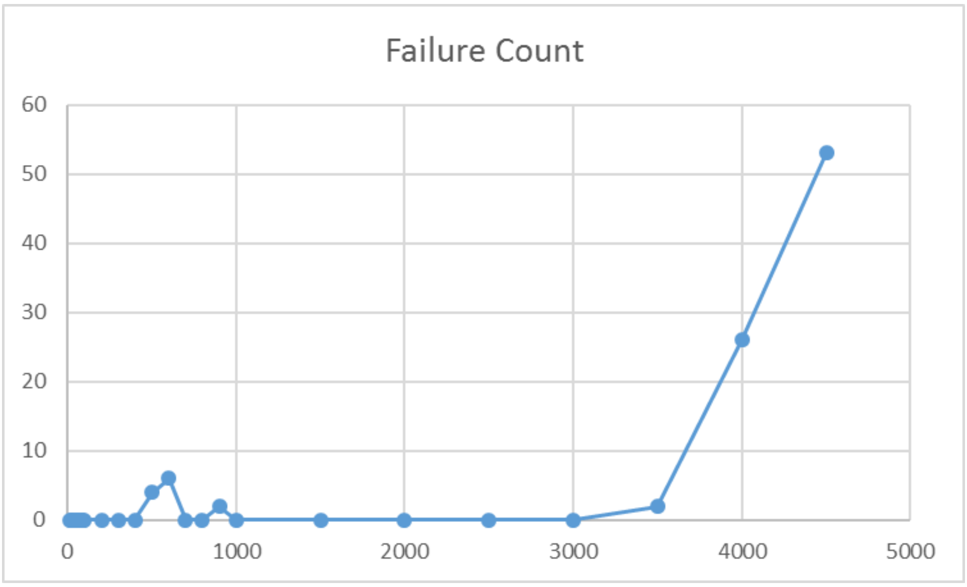


Figure 18: Multi Node Failure Count

Multi Node Vs Single Node Request Time Variance

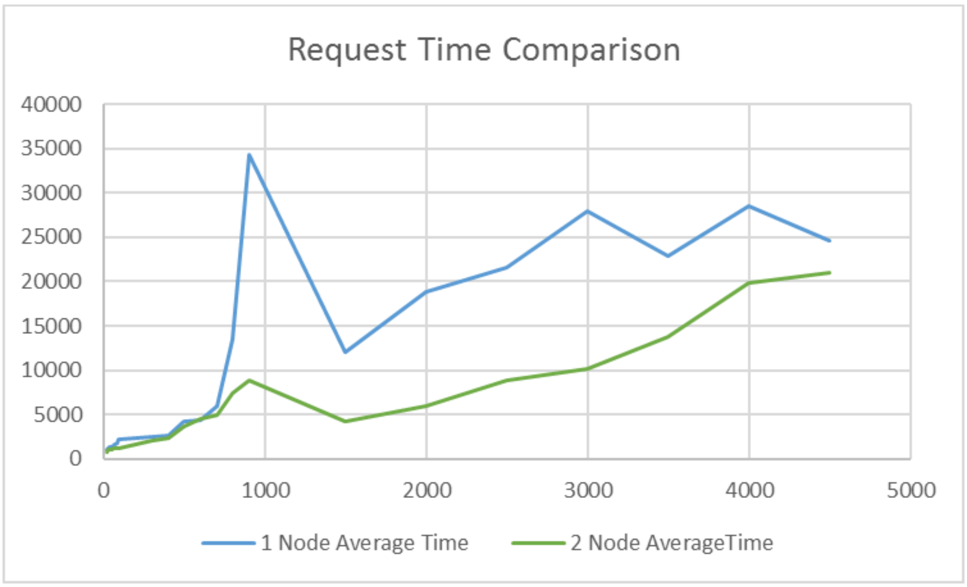


Figure 19: Single Node vs Multi Node Request Time Processing Comparison

Chapter 5
Conclusion and Future Work

An augmented UCON model for e-healthcare system has been developed which overcomes the shortcomings of existing implementation of this access control model. Proposed model has features of high availability, resilience, high performance and security. Sun's XACML policy server along with Microsoft Cloud Azure has been used for the implementation of proposed model. All features of UCON related to authorization, obligation, condition, continuous evaluation and attribute mutability have been incorporated.

As part of future work we tend to work more towards the data security. Encryption of data that is being stored and transmitted will be explored further. There are many existing algorithms that have used for data encryption in other models like Attribute Based Access Control, Role Based Access Control. Analysis of current encryption algorithm, feasibility to implement it in UCON and any enhancements to it will be part of future work.

References

[1] Lili Sun and Hua Wang. A Purpose Based Usage Access Control Model for EHealthcare Services, IEEE, 2011

[2] Tina Tanvi, Mehdi Shajari and Peyman Dodangeh. A usage control based architecture for cloud environments, IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum, 2012

[3] Arlindo Luis Marcon Jr., Altair Olivo Santin, Maicon Stihler, and Juliana Bachtold Jr. A UCON_{ABC} resilient authorization evaluation for cloud computing, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, 2014

[4] Mounira Mshali and Ahmed Serhrouchni. Access control in probative cloud, The 8th International Conference for Internet Technology and Secured Transactions (ICITST-2013), 2013

[5] Dongliango Jiao, Liu Lianzhing, Li Ting and Ma Shilong. Realization of UCON model based on extended XACML, International Conference on Future Computer Sciences and Application, 2011

[6] Fang Pu, Daoquin Sun, Qiying Cao, Haibin Cai and Fan Yang. Pervasive computing context access control based on UCON_{ABC} model, International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP'06), 2006

[7] Yonggang Ding and Junhua Zou. DRM Application in UCON_{ABC}, Advanced Software Engineering & Its Applications, 2008

[8] Tamleek Ali, Mohammad Nauman, Fazl-e-Hadi, Fahad bin Muhaya. On Usage Control of Multimedia Content in and through Cloud Computing Paradigm, IEEE, 2010

[9] Fengying Wang and Fei Wang. The Research and Application of Resource Dissemination Based on Credibility and UCON, International Conference on Computational Intelligence and Security

[10] Zhang Guoping and Gong Wentago. Research of access control based on UCON in the Internet of things, JOURNAL OF SOFTWARE, VOL. 6, NO. 4, APRIL 2011

[11] Jaehong Park and Ravi Sandhu. The UCON_{ABC} Usage Control Model, ACM Transactions on Information and System Security, Vol. 7, No. 1, February 2004