Universidade de Aveiro Departamento de Electrónica, Telecomunicações e
2015 Informática

RITIKA THAKUR

Access Control Model to Support Orchestration Of CRUD Expressions

Modelo de Controlo de Acesso para Suportar Orquestração de Expressões CRUD

Universidade de Aveiro

Departamento de Eletrónica, Telecomunicações e Informática

2015

RITIKA THAKUR

Access Control Model to Support Orchestration of CRUD Expressions

Modelo de Controlo de Acesso para Suportar Orquestração de Expressões CRUD

Tese apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Computadores e Telemática, realizada sob a orientação científica do Doutor Óscar Mortágua Pereira, Professor auxiliar do Departamento de Eletrónica,Telecomunicações e Informática da Universidade de Aveiro

I would like to dedicate this thesis work to my family and to Ajay Kaushal

o júri / the jury

presidente / president	Prof. Doutor André Ventura da Cruz Marnoto Zúquete professor Auxiliar, Universidade de Aveiro
vogais / examiners	Prof. Doutora Maribel Yasmina Campos Alves Santos professora Associada Com Agregação, Universidade do Minho
orientador/supervisor	Prof. Doutor Óscar Narciso Mortágua Pereira, professor Auxiliar, Universidade de Aveiro

acknowledgement

I would take this opportunity to thank my family for supporting me and giving the courage to finish this work. I am really grateful for the guidance and support provided to me from Prof. Doutor Óscar Narciso Mortágua Pereira in the completion of this work. In the end I would like to thank everyone who helped me during the progress of the work.

palavras-chave controlo de acesso, segurança de informação, bases de dados, expressões CRUD, orquestração

resumo O controlo de acesso é um aspecto sensível e crucial quando se fala de proteger dados presentes em base de dados. Em aplicações que assentam numa base de dados baseadas em expressões *Create*, *Read*, *Update* e *Delete* (CRUD) , os utilizadores podem executar uma ou uma sequência de expressões CRUD para obter um dado resultado. Neste tipo de aplicações o controlo de acesso não é limitado apenas a autorizar o acesso a um objecto por um sujeito, mas também a autorizar e validar as operações que o sujeito pode fazer sobre os dados depois de obter autorização. Os modelos atuais de controlo de acesso geralmente focam-se em restringir o acesso aos recursos CRUD a CRUD. No entanto, logo que o sujeito é autorizado, não há restrições sob as ações que este pode efetuar sobre esses recursos. Neste trabalho é apresentado um modelo de controlo de acesso que extende as funcionalidades dos modelos de controlo de acesso atuais para fornecer um ambiente onde um conjunto de politicas predefinidas são implementadas como grafos de expressões CRUD. Estes grafos de expressões CRUD são considerados como sequências que atuam como politicas guardadas e preconfiguradas. O design das sequências é baseado nas operações que o utilizador deseja efetuar para obter um dado resultado. Estas sequências de expressões CRUD são assim usadas para controlar e validar as ações que podem ser efetuadas sobre a informação armazenada. De forma a reusar estas políticas, o modelo apresentado define o uso de execuçao externa de políticas configuradas. O objetivo do trabalho nesta tese é fornecer uma estrutura que permite aos utilizadores de aplicações apenas executarem sequências autorizadas de expressões CRUD numa ordem predefinida e permitir aos administradores de sistema de desenharem politicas de uma forma flexível através de estruturas de grafos. Como prova de conceito, o modelo Role Based Access Control (RBAC) foi tido como referência para o modelo de controlo de acesso e para a base deste trabalho foi escolhido o S-DRACA que permite sequências de expressões CRUD de serem executadas por ordem.

keywords

access control, CRUD expressions, information security, databases, orchestration

abstract

Access Control is a sensitive and crucial aspect when it comes to securing the data present in the databases. In an application which is driven by Create, Read, Update and Delete (CRUD) expressions, users can execute a single CRUD expression or a sequence of CRUD expressions to achieve the desired results. In such type of applications, the Access Control is not just limited to authorizing the subject for accessing the object, but it also aims to authorize and validate the operations that a subject can perform on the data after the authorization. Current Access Control models are generally concerned with restricting the access to the resources. However, once the subject is authorized, there are no restrictions on the actions a subject can perform on the resources. In this work an Access Control Model has been presented which extends current Access Control model's features to provide an environment where a set of predefined policies are implemented as graphs of CRUD expressions. The design of the access control policies is based on the CRUD expressions that a user needs to execute to complete a task. These graphs of CRUD expressions are hence used for controlling and validating the actions that can be performed on authorized information. In order to reuse the policies, presented model allows the inter execution of the policies based on some predefined rules. The aim of the present thesis work is to provide a structure which allows the application users to only execute the authorized sequences of CRUD expressions in a predefined order and allows the security experts to design the policies in a flexible way through the graph data structure. As a proof of concept, Role based Access Control model (RBAC) has been taken as a reference access control model and the base for this work is chosen as Secured, Distributed and Dynamic RBAC (S-DRACA) which allowed the sequence of CRUD expressions to be executed in single direction.

Contents

Table of Figures

List of Tables

List of Acronyms

CRUD	Create, Read, Update, Delete
RDBMS	Relational Database Management System
DBMS	Database Management System
MAC	Mandatory Access Control
DAC	Discretionary Access Control
RBAC	Role Based Access Control
ABAC	Attribute Based Access Control
S-DRACA	Secured Distributed and Dynamic RBAC
SQL	Structured Query Language
SSMS	SQL Server Management Studio
FSM	Flexible Sequence Manager

1 Introduction

Privacy, security and trust are crucial for any application, service and transaction offered over a public communications network, such as the Internet (Rodica Tirtea,2011).The evolution of today's business and technical world are imposing growing demands for flexibility, efficiency, availability and reliability of information upon database applications. The needs and demands of the current business environment are leading us to a scenario where reliability and availability of information from various sources is a crucial issue. In every genera of business processing, storing and securing the information are the most crucial and important tasks in hand. Current business world forces limits to maintain a balance between the securing the valuable information and providing an ease for accessing this stored information. Security Managers design policies which are a collection of principles and rules that describe how an organization intends to protect the confidentiality, integrity, and availability of its systems and the information that they process(Caulfield,2015).

An important requirement of any information management system is to protect data and resources against unauthorized disclosure (secrecy) and unauthorized or improper modifications (integrity), while at the same time ensuring their availability to legitimate users (no denials-of-service). Enforcing protection therefore requires that every access to a system and its resources be controlled and that all and only authorized accesses can take place. This process goes under the name of access control (Pierangela Samarati, 2001).

In today's technology oriented worlds, hundreds of applications access the database either for reading the stored information, modifying existing information, deleting the unwanted information or for inserting new information. These operations are termed as CRUD expressions (Create, Read, Update, and Delete) which symbolises the Insert, Select, Update and Delete operation on the databases. Providing a secured environment where users can execute the CRUD expressions without violating the security policies has always been a challenge to the security experts. Security experts implement the access control models in an organization by determining the allowed activities of legitimate users, mediating every attempt by a user to access a resource in the system(Vincent C. Hu, 2006). Access Control Policies are designed to act as rules for allowing only legitimate users to execute the CRUD expressions on the information which they are allowed to

access. Designing the access control policies depend on the model to be implemented in an organization and on the level at which it will be implemented (ROGER NEEDHAM, 2008).

Ordering the execution of CRUD expression is required where the execution of a CRUD expression depends on the result of the execution of another CRUD expression. To understand the importance of the ordering of the executions of CRUD expressions consider the example of a bank application for transferring money from one account to other. Figure 1 shows SQL table of the current amount of money a client has in his account and Figure 2 shows the SQL table for maintaining the information of the transactions a user performs.

Client_ID	int
Client_Name	nvarchar(50)
Client_Address	nvarchar(150)
Current_Balance	money

Figure 1 : SQL table for Client's Balance

In the table shown in Figure 2 the client can only transfers the money into the account of another client if the amount that he desires to transfer is less than or equal to the balance he has in table shown in Figure 1.

Column Name	Data Type
Transaction_ID	int
Client1_ID	int
Transacted_Value	money
Client2_IBAN	nvarchar(100)

Figure 2: SQL table for transactions carried out by the client

To implement the logic of transferring the money from one account to another Figure 3 shows a piece of java code that implements a method that will take the ids of the two clients and the amount that the client wants to transfer as parameters and performs the operation of transfer if the balance in client's account is more than or equal to the money he is willing to transfer.

```
public void transferMoneyTo(int ClientID, String Client2_IBAN, float amount_Transfer)
        throws SQLException {
    float balance;
    PreparedStatement ps = conn.prepareStatement("Select Current_Balance "
            + "from Client_Information where Client_ID=@id");
    ps.setInt(1, ClientID);
    ResultSet rs = ps.executeQuery();
    balance = rs.getFloat("Balance");
    if (balance > amount_Transfer) {
     ps= conn.prepareStatement(" insert into Client_Transaction(Client1_ID,Transacted_Value"
             +  "Client2_IBAN)Values(@id1,@money,'@Iban') ");
     ps.setInt(1, ClientID);
     ps.setFloat(2, amount_Transfer);
     ps.setString(3, Client2_IBAN);
     ps.executeUpdate();
    } else {
    System.out.println("Not enough funds, your current balance is "+balance);
    }
}
```

Figure 3 Java code implementing the transfer method

In this situation the *select* query for getting the balance of the client has to be executed first as the application needs to check the balance of the client before making a transfer. In this situation if the order of the execution of CRUD expressions is changed, then the transfer will be made without considering the fact that client can have his balance less than the amount he is willing to transfer. Hence we can see that if the order of these two queries is changed the result will be the very different from what is being desired.

Therefore we can conclude that, in systems where the execution of the CRUD expression of one state lead to another state, order of the execution of CRUD expressions is very important. In the example shown in Figure 3, execution of the *select* took the system to a state where the decision of making a transfer is to be done.

Current popular access control models such as RBAC, DAC, MAC provides complete access to data after the user is successfully authenticated (Pierangela Samarati, 2001), but they lack in providing a control over the execution of the authorized CRUD expressions that authenticated user can perform on the information source (Ausanka-Crues, 2006). This means that once a user is granted permission to access the secured

information, he is able to perform authorized operations in any order as desired by him. Consider the example depicted by Figure 3, if there is no control on the order of the execution of the CRUD expressions then the application can execute the *insert* query before the *select* query which will lead to a failure and hence an exception will be thrown.

In critical database applications where the system is growing in size and complexity, users executes the sequences of CRUD expression to achieve a particular goal. These sequences either comprise of a list of CRUD expression or a single CRUD to complete a task in hand. In such applications, the access control has to be maintained at sequence level instead of controlling the execution of each CRUD individually, as the successful execution of the sequence results in achieving the goal. Example shown in Figure 3 illustrates the importance of controlling the execution of the sequence rather than controlling the execution of the CRUD.

To overcome this security gap, we provide a structure where an access control policy can be designed as a directed graph of CRUD expressions for validating the actions performed by an authenticated user. This directed graph of CRUD expressions is regarded as policy graph.

Reusing a piece of code written for a specific task is very powerful feature of many programming languages such as Java, C#, php etc. Figure 4 shows an example where getClientBalance() gets the balance of the client using the SQL *select*. The access control policy designed for this method will only allow the execution of the *select* query. Figure 5 shows a modified version of the java code shown in Figure 3, here the method getClientBalance() is being called to execute the *select* query. Figure 6 shows another method to deposit money into a client's account using the same function shown by Figure 4.

To design the access control policy that validates the execution of sequence of CRUD expressions shown in Figure 5 and Figure 6 we need to use the access control policy designed for the execution of sequence of CRUD expressions for getClientBalance() shown in Figure 4. Therefore the policy designed for validating the *select* query shown in Figure 4 is used by the policies designed for validating the respected *insert* and *update* queries.

```
public float getClientBalance(int ClientID) throws SQLException {
    float balance;
    PreparedStatement ps = conn.prepareStatement("Select Balance "
            + "from Client_Information where Client_ID=@id");
    ps.setInt(1, ClientID);
    ResultSet rs = ps.executeQuery();
    balance = rs.getFloat("Balance");
    return balance;
}
```

Figure 4: java code for getting the balance of a client

If we consider designing a single policy graph to control the order of execution of the CRUD expressions present in the methods of transferring and depositing money, then we eliminate the possibility of reusing the policies. Therefore if policy is designed just to support the CRUD expressions for getClientBalance(), then it can be reused by the policies designed to support transferMoneyTo() and depositMoney(). Hence security manager can reuse the designed policies in different scenarios.

```
public void transferMoneyTo(int ClientID, String Client2_IBAN, float amount_Transfer)
      throws SQLException {
    float balance;
    PreparedStatement ps;
    balance = getClientBalance(ClientID);
    if (balance > amount_Transfer) {
        ps = conn.prepareStatement(" insert into Client_Transaction(Client1_ID,"
                + "Transacted_Value"
                + "Client2_IBAN)Values(@id1,@money,'@Iban') ");
        ps.setInt(1, ClientID);
        ps.setFloat(2, amount_Transfer);
        ps.setString(3, Client2_IBAN);
        ps.executeUpdate();
    } else {
        System.out.println("Not enough funds, your current balance is " + balance);
    }
}
```

Figure 5: Modified java code for Figure 3

```
public void depositMoney(int ClientID, float amount) throws SQLException {
    float balance;
    balance=getClientBalance(ClientID);
    balance=balance+amount;
    PreparedStatement ps= conn.prepareStatement("Update Client_Information "
            + "set Current_Balance=@balance "
            + "where Client_ID=@id ");
    ps.setFloat(1, balance);
    ps.setInt(2, ClientID);
    ps.executeUpdate();

}
```

Figure 6 : java code using method shown in Figure4 for deposit

Figure 5 shows the method transferMoneyTo() which uses the result of the *select* query to make the decision of next query to be executed. So, the result of the *select* query will decide in which state the system will enter: either it will enter into the state where the money is transferred or it will tell the client that he has insufficient funds. The scenarios where execution of a CRUD decides the state that the system will enter requires access control policies that should limit the user to execute only the CRUD expressions which are allowed in resulting state. Design of such access control policies can map the implementation of the applications to the designing of the policies.

In database applications the sequence of CRUD expressions are designed to achieve a goal (Óscar Mortágua Pereira, 2014). The execution of the sequence as a single unit is important to complete the task in hand, if any of the CRUD expressions fail to execute then the system should consider the failure of the task in hand. An access control policy validates the execution of the CRUD expressions but the decision of rolling back the execution of the CRUD expressions in case of failure of execution is made by the application itself.

The current thesis work demonstration will give a direction and solution where the security managers can design these policies as policy graphs and can reuse the policies in different scenarios.

For reference Role Based Access Control (RBAC) model has been chosen as it is one of the most widely trusted and deployed access control model where access to data

is based on the role of a person (Kangsoo Jung, 2013)(D.Richard Kuhn, 2001) and the base of this work is S-DRACA which allows the execution of CRUD expressions in a single direction only (Óscar Mortágua Pereira, 2014).

This Chapter is divided into four sections. Section 1.1 explains the problem which is addressed by this work, section 1.2 explains the solution proposed to solve the problem explained in problem formulation, section 1.3 explains the contribution of this work to current state of the art, section 1.4 brings light on technologies being used in the development of this work and section 1.5 explains structure of the presented work.

1.1 Problem Formulation

Popular access control models such as RBAC, DAC, MAC etc. provide an environment where the security of the information is just limited to the authentication of the user and authorizing the access to the secured resources. These access control models lack in providing the control over action performed by the user on the secured information (Ausanka-Crues,2006). Previously reported work S-DRACA, was able to provide a secured structure where a user was restricted to execute the allowed CRUD expressions for a particular role in one direction (Óscar Mortágua Pereira,2014; Óscar Mortágua Pereira, 2015).

However, it lacks in providing the flexibility that a security manager can have while designing the policies. It allowed the execution of the CRUD expressions in single direction which limits the security managers to design the policies following a uni-directed path. Moreover security experts could only design the policies for the system where execution of a CRUD expression results into one possible state and system cannot enters back into its previous state.

To address the problems elaborated in the introduction section and the limitations of S-DRACA (Secured, Distributed and Dynamic RBAC for relational applications) and RBAC (Role Based Access Control), this thesis work presents a model which combines the access control provided by current access control models with an environment where the security experts can design the policies in the form of policy graphs.

As in database application the sequences of CRUD expressions are executed to achieve a particular task, therefore policies must be defined to act on the sequence level rather than acting on the execution of each CRUD expression (Óscar Mortágua Pereira, 2014). If any of the CRUD expression fails to execute during the life cycle of the policy

graph then depending on the design of the policy continuing or discarding the execution of CRUD expressions is decided.

Presented work also supports halting the execution of a policy graph by starting a new policy graph which is allowed for the user. The reason for including this feature into the model is to support the concept of one method calling another method in a program, and with the addition of this feature to the model the predefined policies can be reused in different scenarios.

The model given by this work defines the rules that should be followed while designing access control policies as policy graphs, and the conditions that should be met when a policy graph is halted to start the execution of another sequence validated by another policy graph.

Therefore motivations for this work can be summarized as follows:

- The security gap in the current Access Control Models, where the user actions are not controlled after he is authorised access rights to a valuable resource.
- The requirement of having control over the sequence of CRUD expressions rather controlling the execution of single CRUD, as database driven applications require a sequence of operations to complete a task.
- The need of flexibility in the design of the security policies, which is required by the security experts while mapping the real world implementation of applications to the policies design.

1.2 Proposed Solution

Based on S-DRACA which is Secured Distributed Dynamic RBAC model (2.2), this thesis work gives an access control model which supports designing the access control policies in the form of sequences of CRUD expressions by implementing the structure using directed graphs. The Model introduces the flexibility in the design and execution of the policies. This work uses RBAC model as reference model by authenticating the clients based on the roles they are assigned, but after the client is authenticated, presented model control the operations a client can perform on the database by validating the action against the policies which are implemented as graph structures.

Proposed solution works in an environment where the databases support the execution of CRUD expressions. If the underlying data source does not support the execution of CRUD operation the model cannot be applied to it as the design of the security policies is implemented by graph structures of CRUD expressions regarded as policy graph. The access control policies are stored in the database and when they are requested then they are structured as directed graphs.

Designing the policy is made flexible by using graph structure approach where the execution on one CRUD expression can lead to one or more options of CRUD expressions that can be executed. To provide a solution that meets the demands of complex database applications, halting of the policy graphs is being permitted by storing the state of the system in which the execution of one sequence was paused to start the new sequence.

1.3 Contributions

This works aims to provide an Access Control Model that can be applied with the current access control model such as RBAC, DAC, MAC etc. and it provides the control over the execution of the CRUD expressions that a user can execute. Presented model provides the rules for designing the access control policies and the rules that should be followed when a policy graph used for validating the execution of the sequence of CRUD expressions is halted to start the execution of a new policy graph. This works tries to map the real world application implementations into the design of the policies.

The presented work allows the policies to be written in the form of directed graphs of CRUD expression which can utilize the access control provided by the other directed graphs of CRUD expression. This feature allows the security experts to design policies in an efficient way as the policies can be reused in different scenarios.

1.4 Tools and infrastructure used

Java is being used to implement the various components of the model, which gives the work advantage of being portable. The standard libraries for Java Serialization and Deserialization allows sending and retrieving objects to and from remote machines, and this feature was used to send the policies as graph structure objects to the client side and on the client side these objects were deserialized to obtain the predefined structure of the policies(Oracle Java Documentation,2015).

NetBeans IDE is being used to develop different components of this work. This IDE provides the environment where the web applications, desktop applications and mobile applications can be designed, code can be indented, a specific word can be easily found in hundreds of lines of code and code can be refracted too(Netbeans IDE features,2015).

The predefined policies are stored in the relational database using Microsoft SQL Server. SQL Server Management Studio (SSMS) 2012 is being used to design the queries for storing and retrieving the data from the database. SSMS provides an environment where all the components of SQL Server can be accessed, managed, configured and developed(SQL Server Mangement Studio,2015).

Jgrapht library is being used to create the graph structures which represent the predefined stored access control policies. As the graph structures are extensively used in this project, so using Jgrapht library for implementing the structure of the policies reduced an overhead of implementing method for designing the graphs. This library provided the functions such as adding the node, adding a directed edge, adding loops, adding weighted edges etc. It supports various types of graphs such as directed, undirected, unmodifiable and listenable etc(Jgrapht,2015)

1.5 Structure of the Dissertation

Thesis structure is categorised into chapters and these chapters are further subcategorised into sections. Chapter 2 is State of the Art which elaborates the work done in the area of access control, Chapter 3 is Technological Background which brings light on the various technologies used in the building of the project, Chapter 4 is Access Control Model for the orchestration of CRUD expressions which describes the presented model, the rules for constructing the model and the proof of concept, the final Chapter 5 is the Conclusion and future work.

2 State of the Art

This section gives an overview of the current state of proposed thesis work based on access control policies, databases and distributed applications. The section includes the discussions on the access control policies, Secured Distributed, Dynamic Role Based Access Control (S-DRACA) and the recent related works in the field of access control. An insight of how presented model tries to create various components based on the ideas from them has also been presented.

2.1 Access Control Policies

In this section we discuss the current access control policies. Access to valuable sources is restricted by defining the set of rules known as policies in order to allow only authorized user to access authorized source. The objective of designing these policies is to make the resources only available to legitimate users by restricting unauthorized access attempts. The most important requirements that must meet are integrity, availability and secrecy. An important requirement of any information management system is to protect data and resources against unauthorized disclosure (secrecy) and unauthorized or improper modifications (integrity), while at the same time ensuring their availability to legitimate users (no denials-of-service) (Pierangela Samarati, 2001)

To protect the data and resources from unauthorized access, presented work inherits the security layer of S-DRACA which provides the feature of authentication and encryption. This security layer has not been altered by any means in this work; therefore the secured channel provided by S-DRACA is inherited too. In order to protect a resource, rules are defined as policy graph and then these policies are being used to validate the actions of the client. The development of an access control mechanism normally follows three distinct phases as, (i) the definition of the access control policies; (ii) the creation of the security model to be followed and (iii) the definition of the access control enforcement mechanism (Pierangela Samarati,2001).

Before getting into the details of the access control models, there is a need to understand the two entities Object and Subject. Object is the resource that is being accessed by an authorized entity. Subject is the entity that has permission of accessing Object. According to the requirements and the needs of the system, access control policy can be formalized in categorise of:

- Discretionary Access Control
- Mandatory Access Control
- Attribute Based Access Control
- Role Based Access Control

The details of these four classes of access control policies have been further discussed in subsections 2.1.1 to 2.1.4.

2.1.1 Discretionary Access Control (DAC)

Discretionary Access Control authorizes the access to an object solely based on the identity of the accessing entity and the access rights specified for that identity on every object present in the system. Users who are given privileges of accessing the objects can pass their privileges to other users, where as granting or revoking the privileges is done by an administrative authority. DAC policies are what commercial operating systems typically enforce. Here, the principals are users; the objects include files, I/O devices, and other passive system abstractions.

DAC can be implemented by associating an Access Control List to the objects, and these ACL define what are the access right for that particular object and who can have access to it. Systems which implement the DAC policies have access rights equal to NONE until they are defined by a policy. The assignment of privileges using a DAC model can also be implemented via a matrix form where each user has a row and each object has a column. Access Matrix was proposed by Lampson for the protection of resources within the context of operating systems, and later refined by Graham and Denning, the model was subsequently formalized by Harrison, Ruzzo, and Ullmann (HRU model), who developed the access control model proposed by Lampson to the goal of analyzing the complexity of determining an access control policy (Pierangela Samarati, 2001).

Table 1 shows an access control matrix for the DAC system specifying the access rights of different users in a file system. This policy structure is being implemented using matrix for a file system.

Table 1 Access Control Matrix for DAC

Subject\Object	File_1	File_2	File_3	File_4
Subject_1	Read/Write	Write	Own/Read/Write/Execute	-
Subject_2	-	Read/Execute	-	-
Subject_3	Read/Execute	-	Read/Execute	-
Subject_4	-	Write	-	Read

As per the matrix shown in Table 1, the access rights and the owner of the objects can be defined as access control policies for the DAC model implementation. Implementing the policies through the access matrix will require a excessive disk space; therefore this matrix can be implemented using ACLs where each object will carry a list specifying the access control rights for each subject. Authorization tables can also be used to implement the policies defined by the DAC Model. Table 2 shows an example of authorization table for Subject 2 (shown in Table 1) indicating separate columns for the subjects, objects and for the access right. The formation of this table will eliminate the empty spaces that was available in Access Matrix by specifying each access right individually. However, one of the drawbacks of the DAC is that it doesn't control the distribution of the information once being accessed by the legitimate user. For example Subject_1 shown in Table 1 can read File_1 and can write it to File_2 whereas Subject_2 now can read and execute the contents of File_1. This is a big security issue.

Table 2 Authorization Table for a particular subject

Subject	Access Right	Object
Subject_1	Read	File_2
Subject_1	Execute	File_2

2.1.2 Mandatory Access Control (MAC)

MAC is a type of access control that allows a central entity to constrain the ability of a subject. A central authority defines the policies to deny or allow the access to the resources. In comparison to DAC, the owners of the information are not entitled to have the privilege to change the policies in MAC mechanism. MAC aims to define the policies which can be implemented for an entire system rather than designing the policy for a part of the system. In MAC model each object and subject in the system is assigned a security level based on which the access control enforced. The security level associated with the object defines the importance of the information contained in that object and the loss that can happen if that information is leaked to an unauthorized subject. Whereas the security level for the subject is the level of trust that system has on the subject for not leaking the secured information contained in the object to unauthorized entity (Samarati, 1994).

Allowing access to a resource depends on the security level of subject that is trying to access the object. Processing of the request takes into consideration the importance of the secured information and the trust in the entity trying to access the information. The security level of the subject is compared to the security level of the object and based on the relationship between the security levels the decision of granting or denying the permission is made.

There are the two principles that are required to be true while implementing the MAC model in a system supporting the MAC policies as follows:

- Read Down: Subject's security level must be at least as high as of the object it is about to read. This allows only trusted subjects to read the data.
- Write Up: Subject can only write to an object if and only if Object has the security level as that of the Subject or higher than the Subject. This principle ensures that only trusted subject can write to the object.

Therefore the subjects who are of high security level can read the information from a file but cannot write it to the files which are at lower security levels making the information unavailable to the subjects of lower security level. All this is being achieved by following the above two mentioned principles.

MAC introduces the levels of security in a hierarchical form which in military terms or in generic terms can be classified as Top Secret (TS), Secret (S), Confidential (C) and

Unclassified (UC). Therefore if the object is classified as S, then a subject who has security level of S or lower can only write to the object, whereas the subject who has the security level at least S or higher can read from this object. Write Up principle can damage the information of an object with higher security level as the subject with lower security level will not be able to read what is being written and an over write can be obtained (Pierangela Samarati, 2001), (Amanda Crowell).

Figure 7 shows the MAC hierarchical flow of information in the system corresponding to security levels assigned to the object and subject in a system.

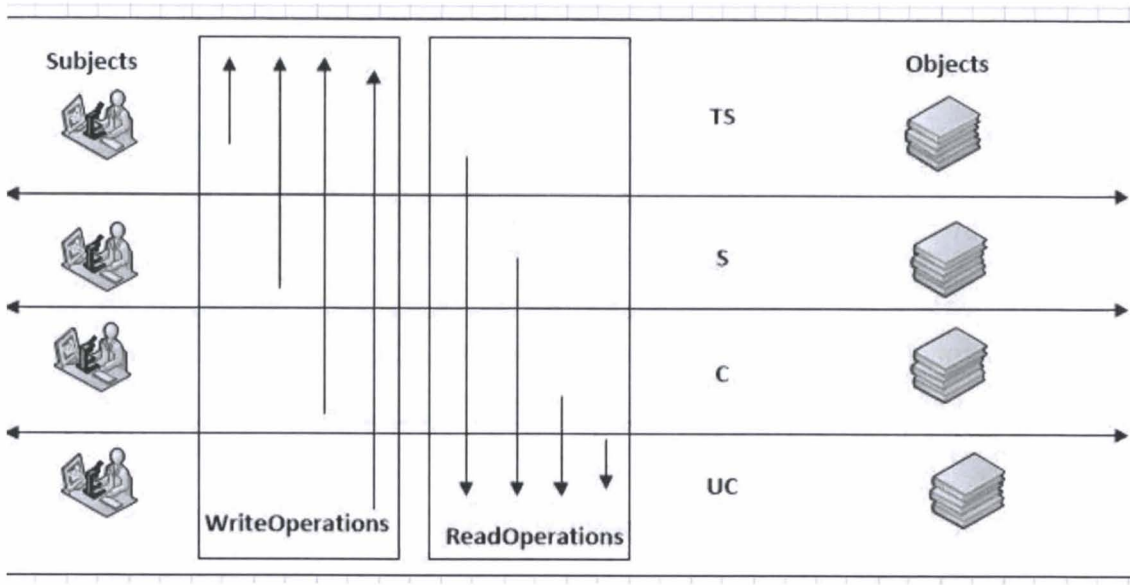

Figure 7 MAC hierarchical flow of information

Mandatory Access Policies for secrecy can also be understood by studying the Bell and LaPadulla model which follows two main principles known as No-read-up and No-write-down. Biba Model has shown the implementation of MAC policies for the integrity by following No-read-down and No-write-up rule (Pierangela Samarati, 2001).

2.1.3 Attribute Based Access Control (ABAC)

As the name explains, access control is based on the attributes of the resources to be accessed and the attributes of the entity trying to access them. ABAC is a logical access control model that is distinguishable because it controls access to objects by evaluating rules against the attributes of entities (subject and object), operations, and the environment relevant to a request(Vincent C. Hu, 2014). Since past decade, many security

experts and organizations have started to use ABAC for access control. In December 2011, the Federal Identity, Credential, and Access Management (FICAM) Roadmap and Implementation Plan v2.0 took a step of calling out ABAC as a recommended access control model for promoting information sharing between diverse and disparate organizations(ABAC).

ABAC systems are capable of enforcing both DAC and MAC concepts. Attributes of a subject can be its name, address, date of birth, role in an organization etc. These attributes combined with themselves or with other attributes forms an identity of the subject.

2.1.4 Role Based Access Control (RBAC)

Role Based Access Control (RBAC) is a popular implementation of ABAC, where roles are considered as attributes and based on this attribute access control is implemented. As the name explains itself, it is access control model based on the roles allotted to the entities present in the system. In RBAC entities perform certain sets of operation which are represented by their roles. Current RBAC model doesn't provide the support for controlling the operation performed by the entities after the roles has been assigned. This is a security gap which is tried to reduce with this work.

As role of the entity represents the level of authorization it gets for performing tasks on the valuable resource, therefore it is important to have a separation of duties related with the roles. In separation of duties the level of authority given to a particular role is tried to be restricted. As in real world one person can't perform all the roles in an organization, therefore different categories of tasks are assigned to different roles for the fulfilment of a managed and successful enterprise.

These separate tasks of duty can be categorised as follows:

- Static Separation of Duty: In this, the roles are being assigned to the individuals beforehand and the users perform the transactions according to pre-assigned roles.
- Dynamic Separation of Duty: In this, the roles of the individuals can vary according to the demand of the system. A user can play different roles as the requirement of the system changes at runtime.

Sandhu et al. have discussed these categories in more details by considering a scenario where an owner of a car company is assigned the role of the manager. In static separation of duty the owner can only authorize the production of the cars, hire more employees, take care of the accounts etc. But in case of dynamic separation of duty the owner of the company can be the buyer for a car from the company and the seller too. Dynamic Separation of duty brings flexibility to the system (Ravi S.Sandhu, 1995).

RBAC model implements the policies based on the group of people which can perform a specific task. Taking an example of flourishing access control in a company, a scenario where the employees of the company can be divided into different groups which can perform the set of task. These groups can be seen as separate group of managers, technicians, database handlers etc. This feature of RBAC model allows it to be easily implemented in many scenarios. As it provides the platform where restriction to a resource is based on the task a person performs by utilising that resource.

A role can have an associated set of members. As a result, RBAC provides a means of naming and describing many-to-many relationships between individuals and right (David F. Ferraiolo, 1992).

Figure 8 shows role relationships of the user in an environment where the Role_A is set of two members and Role_B has only one member. In this case, these users are allowed to access two different resources based on their roles assignment. Transactions are being used to show that changes made to the file will either follow the entire rule or none.

It has been reported that when assigning the roles, principle of least privilege is being implemented by the security manager(Ravi S.Sandhu, Edward J.Coyne, Hal L.Feinstein and Charles E.Youman, 1995). This principle states that a user is given no more privilege than it need to access the information for its needs. Ensuring least privilege requires identifying what user's job is, determining the minimum set of privileges required to perform that job and restricting the user to a domain with only those privileges (David F. Ferraiolo, 1992).

A very important concept of RBAC model is role hierarchies. Role hierarchies represent the level of authorization a role holds for accessing a resource. If the role is of administrator, then operations such as Insert, Delete, and Update are being allowed but if the role is of the reader then these operations are not authorized. In organizations

mangers hold the roles to modify the information, but the employees are categorised in roles where they can either insert or read information or can perform both operations.

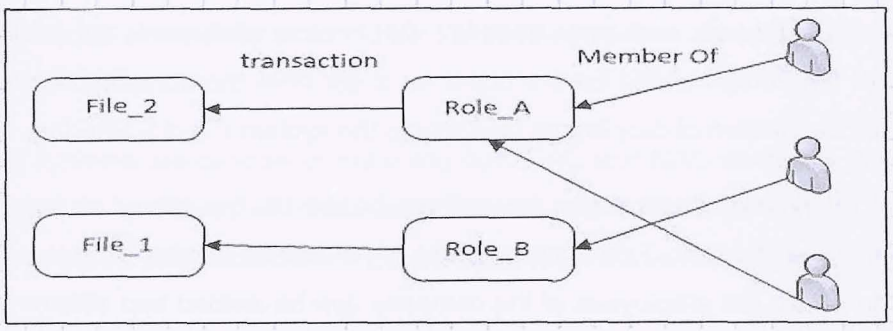

Figure 8 Role Relationship

RBAC is a flexible model as it takes on the real world organizational characteristics and defines them as policies (David F. Ferraiolo, 1992). Figure 9 shows an example of a bank showing Multi-Role relationships of the bank Manager, Accountant and Clerk with different subjects as well as objects. In this figure Manager inherits the membership of the Accountant and Clerk hence has access to the transactions of all the members including accountant and clerk as well as his own. Whereas for accountant, the membership of clerk is being inherited hence has the access to all the transactions of the clerk and his own but not that of the manager. However, the clerk is only allowed to have access to the transactions which he was assigned excluding others.

National Institute Of Standards and Technology proposed RBAC model which was defined in terms of four component models named: (i) Core RBAC; (ii) Hierarchical RBAC; (iii) Static separation of Duty Relations; and (iv) Dynamic separation of Duty Relations (D.Richard Kuhn, 2001). Core RBAC models defines minimum set of RBAC element which are essential to implement an RBAC model such as the user-role assignment and the permission-role assignment relations. However, in Hierarchical RBAC model, the support for role hierarchy was introduced. The user with superior role automatically inherits the permissions assigned to the roles that are lower in hierarchy and roles who are lower in hierarchy automatically inherit the users of roles higher in hierarchy. In Static Separation of Duty Relation model, the exclusivity between the users and the roles assigned to them is being introduced. This can be understood by taking an example of security guard and the clerk in a bank, both have two totally separate roles and therefore the role of security is not to be assigned to the role of the security guard (Virgil D. Gligor,

1998). The Dynamic Separation of Duty Relation model defines the exclusivity between the roles that are activated as part of user sessions.

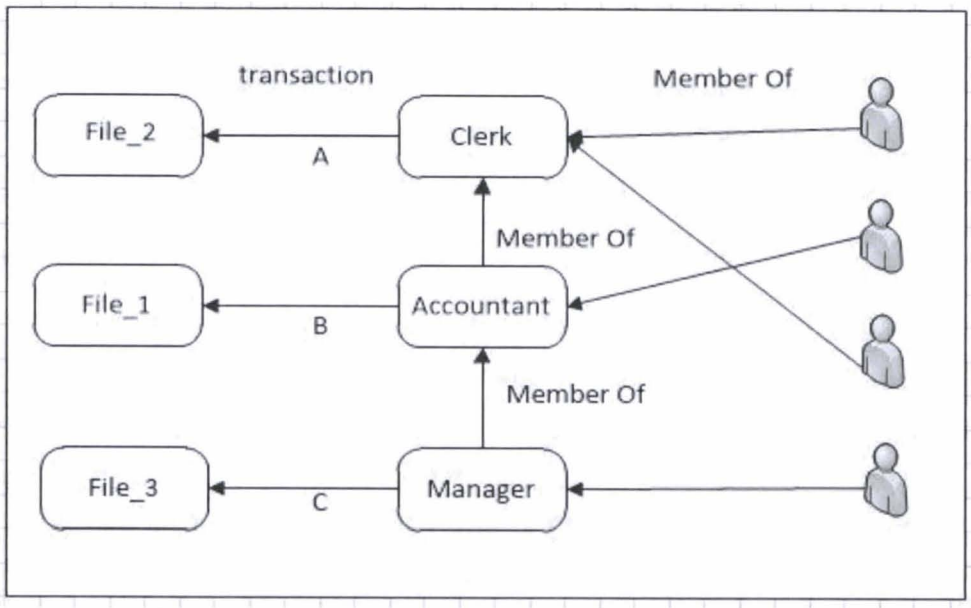

Figure 9 Multi-Role Relationships

In the article entitled "Role-Based Access Control Models" reported by Ravi S. Sandhu (Ravi S.Sandhu, 1995), RBAC has been defined in terms of three different models which show how flexible and generalized RBAC can be. These models were defined on the basis of different aspects of RBAC and how they can be related to different scenarios.

2.2 Secured Distributed, Dynamic Role Based Access Control (S-DRACA)

Secured Distributed, Dynamic Role Based Access Control (S-DRACA) presents a structure which has a security layer providing authentication and a secured channel for communication. It makes an attempt in overcoming the security gap in RBAC model by restricting the execution of the CRUD expressions allowed for a role in one direction(Diogo José Domingues Regateiro, 2014).

S-DRACA provides an easy interaction between the users and relational databases without the overhead of having knowledge of the database schema and access control policies. It provides a platform for the application users where they can design the application without having pre-knowledge of the access control policies.

S-DRACA stores the pre-defined policies as sequence of CRUD expressions which act as permissions for the authorized roles to execute a CRUD operation. These policies are being retrieved by the client side by making a request to the server; an entity on the client side uses these pre-defined policies to validate the execution of CRUD expressions. It enforces the changes made to RBAC policies at run-time, therefore the change in policy on the server side is mapped to the client side and the application can change its behaviour and avoiding the run time exceptions. (Óscar Mortágua Pereira, 2014)

The granular information about the architecture of S-DRACA has been further explained in Chapter 3 section 3.1

2.3 Related Works

As the demand and need of access control has changed over the past few years we have seen many researchers trying to give better models meeting the demands of the different branches of the business world. In this section we discuss in brief some of the works that has been done in the area of access control in the past few years.

2.3.1 Hybrid approach for XML access control (HyXAC)

Mangona Thimma considered the growth of the usage of XML and gave a model named as 'Hybrid approach for XML access control', which enhances the XML access control model (Manogna Thimma, 2015). HyXAC pre-processes user queries by rewriting the queries and eliminating the parts of the queries which are violating the access control rules. In particular, HyXAC firstly pre-processes user queries by rewriting queries and removing parts violating access control rules, and then evaluates the re-written queries using sub-views, if they are available.

In HyXAC, views are not defined on a per-role basis; instead a sub-view is defined for each access control rule. The roles sharing identical rules will share sub-views. Moreover, HyXAC dynamically allocates memory and secondary storage resources to materialize and cache sub-views to improve query performance. We have conducted extensive experiments, and the results show that HyXAC improves query processing efficiency while optimizes the use of system resources. The QFilter approach used in HyXAC for query pre-processing and been claimed to optimize the usage of system resource that alternatively improves the performance of query processing.

2.3.2 JIF

Jif (Java + information flow) is a security typed programming language written in java that extends Java with support for information flow control and access control. It provides access control and information-flow control both at run time and compile time by using the labels (JIF, 2015). The labels describe how the information flow should be used. For example consider a java script label as:

int {Alice -> Bob} x;

The label in the above example explains that information in variable x is controlled by Alice and Alice permits Bob to see the information. Thus one can not only know that the variable x is of type int but also knows about the flow of information explained by the label. However, if the label is described in other direction as:

int {Alice <- Bob } x;

Then that means the information is owned by Alice and Alice permits Bob to make changes to the information contained in variable x. Hence to summarize these label annotations, one can conclude that a Jif compiler analyses information flows within programs to determine whether they enforce the confidentiality and integrity of information. One of the disadvantages of Jif is the lack of libraries those should be translated from Java to Jif to make Jif usable in a larger scale.

Although signatures can permit the use Java libraries, but it still requires to write those signatures. Nevertheless using signatures for Java classes does not provide security because no check is done to verify that the labels of the signatures correspond to the security of the Java classes. For now this has to be done by hand. Moreover, the Jif has been mostly used to manage the information flow at the application level and not to access data in RDBMS, so other tools must be used to compensate(Boniface Hicks, 2007).

2.3.3 Paragon Policy Language

Paragon is an extension to the java language that enables practical programming with information flow controls (Paragon, 2015). In this language the entities those are related to the information flow are described as actors which can be a user, resources, a system component etc. These actors are represented as object references. A program policy is

used to label the information containers (local variables, fields) and specifies to which actors the contained information will flow. For example, the code fragment shown below can creates regular instances of the User and File class, where alice and file_1 can play a dual role; both as program variables, and as actors in Paragon policies.

Actors as object references:

$$User\ bob = new\ User();$$

$$User\ alice = new\ User();$$

$$File\ file\_1 = new\ File();$$

$$File\ file\_2 = new\ File();$$

Paragon Policies:

$$\boldsymbol{policy}\ p1 = \{\boldsymbol{alice}:;\ \boldsymbol{bob}:\};$$

$$\boldsymbol{policy}\ p2 = \{\boldsymbol{Flie\ f}\ :\};$$

A policy consists of a set of items, each specifying a particular actor or a group of actors. For example, policy p1 shown in above code fragment, states that information may flow to specific user alice or bob and policy p2 states that information may flow to any file. This makes the policy {Object o:} the most permissive, and the policy with no clauses (denoted {:}) the most restrictive paragon policy. Paragon also defines the clauses for the states which can restrict the flow of information for the actors (Niklas Broberg, 2013).

When compared to Jif, Paragon has the advantage of having the flexibility of the concept of locks. Paragon can provide different declassifying methods to work on different data in compare to single declassify construct in case of Jif. However, still, much work is left to be done before Paragon can become a serious competitor to existing programming such as Jif. It requires both theoretical and practical work, in particular if declassification mechanisms are shared among threads. A substantial formalisation of Paragon's type system has been lacked so far, including a proof of soundness with respect to information flow security.

2.3.4 Multi-Level Dynamic Access Control Model

Recently, Zhou et al. (Yanjie Zhou, 2015)has formalized a five-level dynamic access control model architecture which describes the relation between access control systems and applications. They used Role-based Access Control (RBAC) as a reference access control model. RBAC model has been described in a five-level architecture followed by web services integration into RBAC model. The reported model distinguishes two kinds of actions in the entire access control systems: the administrative actions and the application action. These actions may cause changes of the access control components and resources. The reported five-level in proposed model are claimed to demonstrate the logic relation between access control systems and applications. It also explains that dynamic property of access control is partly decided by applications.

2.3.5 Graph theory to access control

In 2004, a researcher Ravi Sandhu from George Mason University, USA reported a perspective on the connections between graphs and access control models, particularly with respect to the safety problem and dynamic role hierarchies(Ravi Sandhu, 2004). The work explored a connection between graphs and information security especially in the area of access control and authorization.

2.3.6 Security-driven model-based dynamic adaptation

In 2010, Morin et al. reported a methodology for implementing security-driven applications by demonstrating a model named security-driven model-based dynamic adaptation, reflecting the access control policy (Brice Morin, 2010). The model addresses a problem where even with the separation of the policies and the application code proved in the theory but never fully done is practice, leading to some rules being written directly in the application code. The approach uses meta-models that describe the access control policies and the application architecture. It defines the mapping (statically and dynamically) of the access control policies meta-model to the application architecture meta-model. However, the model lack in addressing how to statically implement secure and dynamic security mechanisms.

2.3.7 Ur/Web

Chlipala et al. (Chlipala, 2015) has demonstrated a simple model for programming the web named Ur/Web, which allows programmers to write CRUD expressions that can check statically the access control policies in a system backed by a DBMS. Ur/Web is a domain-specific, statically typed functional programming language for programming modern web applications. Ur/Web's model is unified, where programs are compiled in a single programming language to other "Web standards" languages (Chlipala, 2010). It supports novel kinds of encapsulation of Web-specific state and exposes simple concurrency where programmers can reason about distributed, multithreaded applications. It allows the programmers to write the CRUD operations to check statically the access control policies present in the DBMS. Programs are developed to check that data involved in the CRUD expressions is accessible through some polices. For policies to vary from user to user queries that check them can use actual data and the extension of the SQL which is based on what information needs to be disclosed to user on the basis of what the user already knows. However, Ur/Web programming documentation requires and is limited to Haskell and ML expertise making it particularly suitable for statically typed functional programming.

2.3.8 Reflective Database Access Control (RDBAC)

In 2008, Olson reported on the Reflective Database Access Control (RDBAC) (Lars E. Olson, 2008). RDBAC aids the management of database access controls by improving the expressiveness of policies. In this model, the CRUD expressions are the privileges in the database itself rather than static privileges defined in the access control lists (ACL). Transaction Datalog given by Bohner et al. (Anthony J. Bonner, 1997)can be used to express the reflective access control policies.

2.3.9 Java EE

Java Enterprise Edition (Java EE) is an extension of Java platform and is the standard in community-driven enterprise software (Oracle Java Documentation, 2015). Java EE is developed using the Java Community Process, with contributions from industry experts, commercial and open source organizations, Java User Groups, and countless individuals. A Java EE application server can handle transactions, security, and management of the components it is deploying, in order to enable developers to concentrate more on the business logic of the components rather than on infrastructure and integration tasks. It

uses the @RolesAllowed annotation to enforce RBAC policies directly at the methods level, controlling the permissions to invoke them. However, it does not identify who is invoking the protected methods, meaning any user on the allowed roles can get access to the protected method.

2.3.10 Annotated objects

Object-sensitive RBAC (ORBAC) is an extension of RBAC that can be used with object-oriented programming languages (Jeffrey Fischer, 2009). ORBAC controls the access at the level of single objects allowing a fine grained control. Unlike some frameworks, it allows developers to write access code knowing if they are violating any access control policy or not through its type system.

In 2010, Zarnett et al. also reported RBAC in Java via Proxy Objects using Annotations (Jeff Zarnett, 2010). Their proposed system creates proxy objects which only contain methods to which a client is authorized access based on the role specifications and hence the potentially untrusted clients that use Remote Method Invocation (RMI) then receive proxy objects rather than the originals. They present solution that can be applied to control the access to methods of remote objects via Java RMI, a framework that allows an application to use objects that exist in a different application, possibly in a different machine.

Fischer et al. presented a more fine-grained access control that uses parameterized Annotations to assign roles to methods, whereas Zarnett and co-workers defined each annotation required for the domain of the application. However, these approaches in contrast to ours, do not ease the access to a relational database since the developers still need to acquire a deep understanding of the database schema and the defined access policies to access database objects.

2.3.11 Access Control Driven by CRUD Expressions

Recently, Oscar Pereira et al. have reported on the RBAC model to provide access control in a distributed environment using the CRUD expressions (Óscar Mortágua Pereira, 2014). The work demonstrate a software architectural model from which static RBAC mechanisms are automatically built, relieving programmers from mastering any schema. The concept of Business Schema was introduced as a set of CRUD operations. These sets of the CRUD operations were implemented as interface which hides the direct

and indirect modes of access. Pareira and co-workers has also extended this work with modification to previous reported results providing access control based on the policies that were driven from the sequence of CRUD operations. (Óscar Mortágua Pereira, 2014). Howerver, the model restricted the user to execute the CRUD operation in a single direction only. The present thesis work focussed on to demonstrate a direction and solution where sequences consist of permissions for executing an operation and the security managers can predefine the sequence in a more flexible way and is an extended modification of earlier reported work (Óscar Mortágua Pereira, 2014).

3 Technological Background

This chapter tries to bring light on the technological components being used in this work. Following section gives an idea about how the various technical components are being used in to achieve the goals of thesis and also give a brief introduction to these components.

3.1 S-DRACA Architecture

In this section we bring light on the base of this work which is S-DRACA(Diogo José Domingues Regateiro, 2014)(Óscar Mortágua Pereira, 2014). S-DRACA is a security model which was designed for applications which are distributed and, based on RBAC model it provides the access control by controlling the flow of execution of CRUD expressions in single direction. S-DRACA gave the concept of **Business Schema** which is basically a model from which source code can be automatically generated to handle CRUD expressions. A Business Schema can have relation to one or many CRUD expressions (Óscar Mortágua Pereira, 2014).This concept has been inherited to the present work. Figure 10 shows the architecture of the S-DRACA.

S-DRACA consisted of client side and server side components, on the server side Policy Manager is an entity which is responsible for retrieving the policies from the database know as Policy Server. These policies are designed by the security experts by defining roles for the users and each user is assigned with set of sequences of CRUD expressions. Whenever there is a change in the policies clients are notified of the change by the Policy Manager. It also authenticates the clients and provides a secured encrypted channel of communication.

Policy Extractor is en entity which is integrated in the client application using java annotation and its purpose if to obtains the policies from the Policy Manager and generate components for access awareness. The Business Manager implements the access control mechanism for the client application by getting the policies from the Policy Manager and it also keeps track of the active sequence for a particular role in an active session. It is Business Manager who authorizes the execution of the CRUD operation of the Sequence and modifies them dynamically in accordance with the Policy Manager.

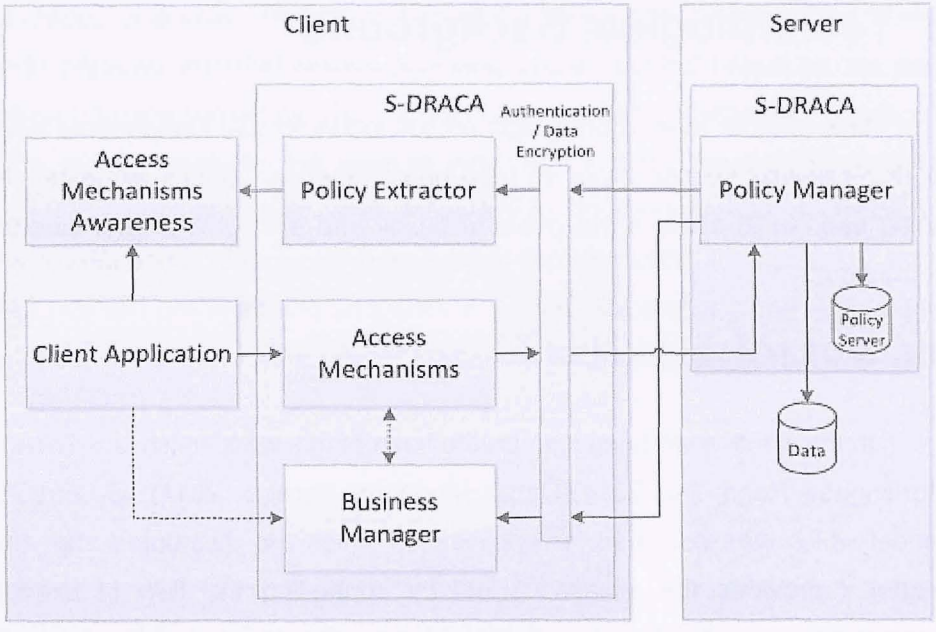

Figure 10 S-DRACA Architecture

Access Mechanism is where the access control policies are being enforced and they inform the client application about the next authorized CRUD operation to be executed by requesting the Business Manger about the next CRUD to be executed in the active sequence. S-DRACA also provides a security layer named as Authentication/Data Encryption which provides a secured communication channel between the client and the server via authentication and encryption.

S-DRACA addressed the security aspect but it lacked in providing the flexibility by restricting security experts to design the policies following a single direction.

3.2 Orchestration

Orchestration in computing describes as an automatic process which arranges, manages and coordinates the complex computer system, middleware and the services. Orchestration is the result of automation.

Orchestration is often discussed in context of systems where one component of the system provides service to another component following some policies. It defines the policies and service levels through automated workflows, provisioning, and change management. The request is sent to the central entity and the reply is provided to the requestor as the service it requested.

In the presented access control model, a user is allowed to execute sequences of CRUD expressions which can be seen as a service which it has authorization to execute. One of the aims of this work is to control the execution of these authorized CRUD expressions. So this works aims to provide a service, which is controlling the execution of CRUD expressions allowed for a particular user. In this model user application request a central entity which is deployed on the client side and this entity acts as a central repository for providing the service of controlling the execution of CRUD expressions and providing the interface for the client application to execute the next authorized CRUD expression.

3.3 Relational Database Management System

Current world demands the speed in data processing and storing. A Database Management System (RDBMS) provides the users, a facility to manage their Relational Databases. In 1990, Edgar F. Codd invented the relational model for database management (Edgar F.Codd, 1990). Database management systems (DBMS) maintains the integrity of the data, provides security of the data, provide data availability and data independency.

Relations Databases store data in the form of tables including rows and columns and the relation between two tables is maintained by having at least one same column values in both tables. To identify the records of the table, a column is maintained where each value is unique known as the Primary key of the table. A Foreign key is the value in a table which identifies the records of another table to which it maintains a relation. Presented work uses the Relational database as a back end for the model. Access Control Policies are being stored in the relational database in the form of tables.

Maintaining the integrity of the data present in the database is one of the important key issues in any organization. For organizations the data is the key ingredient of the work to earn their reputation in the market. In such big organizations the users need to execute a series of steps to fulfil a task in hand. For example, for opening a new user account in a bank, a series of insert operations have to be executed to inert the information into the database and if any of these operations fails, then the information provided in the database will not correct (S. Sumathi, S. Esakkirajan, 2007).

Distributed environment is the need of current business oriented world, which leads us to a situation where different users are trying to access and manipulate the data source

at the same time. This situation brings the database to an inconsistent sate where the integrity of the information is not maintained as several read operation can be executed before the correct data is being written in the database. To avoid such situation DBMS supported locks are being used. The data which is being altered by a transaction during the execution of the transaction is regarded as data element. Lock can be defined as set of permissions for a transaction which allows only authorized operations to be executed for a particular data element. Transaction must get this permission from the Transaction manager before altering the value of the data element (S. Sumathi, S. Esakkirajan, 2007).

Locking protocols are also being used in distributed environment to ensure that result of the execution of each transaction in a disordered pattern will be same as the execution of the transaction in serial order.

DBMS must provide the data to the users in a reasonable format and cost so that the users can easily access the data moreover the access to the data must be authorized to legitimate users. The DBMS acts like a middle layer between the database and the client. It's the DBMS which provide the inter-operability feature to the databases. DBMS hides the complex structure of data storage from the user and just gives an abstract overview.

Structured Query Language (SQL) is one of the most popular programming languages that are being used to write queries for the databases. Users state what information they want from the database or what they want to do to the data of the database via SQL and the DBMS take care of describing data structures for storing the data and retrieval procedures for answering queries.

RDBMS keeps the data consistent by applying the following constraints (Edgar F.Codd, 1990):

- Domain integrity: The domain integrity assures that the value of a particular column is of the column type and whether a null value can be inserted in that column or not.

- Entity integrity: The records in the table are uniquely identity by the Primary key of that table and entity integrity assures that the values of the primary key are not duplicated.

- Referential integrity: in this, the constraints guarantee that the value of a foreign key must exist as the value of the primary key which is being referenced.
- User defined integrity: User-defined integrity allows you to define specific business rules that do not fall into one of the other integrity categories

3.4 Graph Theory for Access Control

This section explains the generic idea behind graphs and how the implementation of graphs in access control has evolved. Graphs are mathematical concepts which are used to represent the relationship between objects. Euler published a paper on graphs which addressed the problem of Seven Bridges of Königsberg and this paper laid the foundation of the graphs. Programmers realized the potential in modelling the real world scenarios as graph structures which lead to the development of many algorithms based on these structures such as Dijkstra, Bellman-Ford etc.

A graph consists of Vertices which are also regarded as nodes and to represent a relationship between these vertices, edges are being used. Graph structures can be directed which means that the direction of the edge between the vertices defines the relationship between the vertices. Undirected graphs are the once in which the edges do not define any direction for the relationship among the vertices. The relationship between the vertices can also be represented as labelled edges.

The source of an edge is regarded as parent node and the destination of that edge if the child node. A node which has no incoming edges is regarded as the root of the graph where as the node which has no outgoing edges is regarded as a leaf node.

Vertices can be any real world entity which can be represented as an object in a programming language. Designing graphs totally depends on the needs of a problem and on the implementation choices of the programmer. Following is the graph structure which is designed to show the access control maintained by Lampson's access matrix:

Table 3 Lampson's Access Matrix

Subject\Object	File_1	File_2	File_3
User_1	RWE	RW	WE
User_2	-	R	W
User_3	Own,RWE	-	-

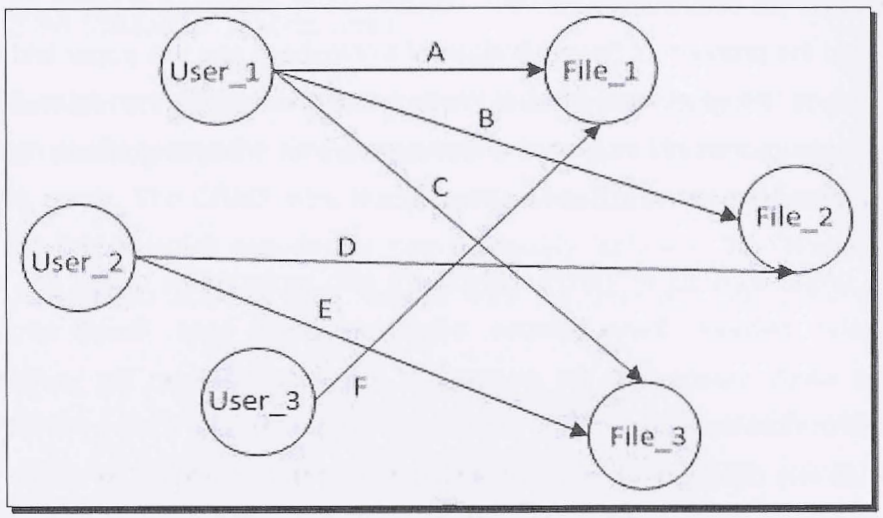

Figure 11 Example of graph representing Lampson's Access Matrix

As we can see in the above example vertices represent the users and the files where as the access to the files is defined by the labelled edges which the user vertices holds to file vertices (Ravi Sandhu, 2004). The label *A* represents the Read, Write, Execute (RWE) permissions, label *B* represents the Read and Write (RW), label *C* represents the Write and Execute (WE) permissions, label *D* represents the Read(R) permission, label *E* represents the Write (W) permission, label *F* represents that User_3 is the owner of File_1 and he has the permission of Read, Write and Execute(RWE) permissions.

Graphs can also be used to analyse the security of the system, one of the example of such graph is Attack Graph. Attack graphs are used to define all the paths which an intruder can use to enter a system and achieve his goals. Security experts use the attack

graphs to detect the threats and to design the measures to eradicate these threats (S. Jha, O. Sheyner , J. Wing, 2002).

This thesis work takes above mentioned approaches into consideration and presents a model where the stored sequence of CRUD operation which acts as access control policies for the users, are formalized into graph structures. Graphs bring the flexibility component by providing the feature of directed edges, which enables the security experts to design the access control policies to meet the demands of the real world. Directed Edges of the graph allows the security experts to design the access control policies to restrict users to follow a predefined path.

3.5 Java

Java is a universally accepted and used language. The popularity of java is based on its features such interoperability, secured, object-oriented and distributed. This thesis work uses java as the programming language to implement the access control model because of the language being portable, object-oriented and distributed.

Following section explains the Java Serialization and Deseralization which is being used in this work to send and retrieve the access control policies. Java Reflection and Annotations are also being discussed in the following sections.

3.5.1 Serialization and Deserialization

Java serialization provides the mechanism where object's data and its type can be represented as sequence of bytes. In distributed environment data needs to be sent from one machine to another and this need is full filled by considering the data to be sent to other machines as objects by using java serialization.

Java provides an interface Serializable which is implemented by the class whose object is to be sent. ObjectOutputStream class is used to serialize an object. The ObjectOutputStream class writes the object as series of byte characters and on the client side these objects are Deserialized by performing read from ObjectInputStream to the same data type and in the same order as they were written. Java Serialization supports refracting which means that adding new fields to the class, changing static fields to non-static fields are automatically managed. A serialization Id is being used to know if the serialized object has been changed or not (Oracle, 2015)(Neward, Ted, 2015).

This work uses java serialization to send the graph structure of access control policies which are defined as sequence of CRUD expressions in the database. These graph structures are defined as object and then Java serialization and ObjectOutputStream is being used to send the policies to the client side. On the client side, these policies are retrieved as graph structures by deserialzing the object using ObjectInputStream readObject ().

3.5.2 *Reflection*

Reflection is generally used by applications which need to monitor or alter the runtime behaviour of a program running in a JVM. It allows a program to know contents of the classes, methods, fields at run time, without knowing their names at compile time.

Reflection is a mechanism present in the Java Virtual Machine that provides the ability of code inspection or modification. For its usage, the Reflection API is provided which can be used inside an application to execute the said abilities. Reflection shouldn't be used without a specific purpose, since for its abilities there are drawbacks as follows:

- **More performance overhead**

Comparing to the normal programming way of class instantiation and method calling, where the compiler will generate runtime code for objects to be directly used, and its methods to be called directly without the need of a second call, Reflection needs to get the correct type dynamically, so it will have to find its class structure at runtime in the classpath, and its methods are called via a second method, which will cause delays at runtime.

- **Security restrictions**

Permission for using Reflection is granted by the Security Manager in the Java Virtual Machine at runtime, thus, if not configured properly can lead the application to terminate prematurely.

- **Security issues**

Reflection can also be used as an alternate way to instantiate a class, which doesn't even needs to be present at compile time, and call it's methods. Special care should be

taken in this technique, because is easy to strip out the class protection mechanisms and not respect them afterwards.

- **Unwanted behaviour**

With the abilities it has, using Reflection can lead to unwanted or unexpected side-effects that, without the compiler's oversee, will be harder to debug.

3.5.3 Annotation:

Annotations are the form of metadata that is being used in java. Annotations don't have direct affect on the operations of the program but are used to provide information to the compiler; they can be used to generate code or XML files (The Java Tuitorials,2015).

The at sign character (@) indicates the compiler that the following text is an annotation. Annotations can be applied to the declaration of a class, field, methods etc. With the release of java 8 now annotations can be applied to the use of types.

A few examples of where types are used is the expression where a new instance of the class is being created (new), when casting is being done, *implements* clauses, and *throws* clauses. This form of annotation is called a *type annotation* (Type annotations and Pluggable type Systems, 2015).

Java 8 release doesn't provide a type checking framework but it allows writing modules for type checking which can be used with java compiler. Following example shows the use of Annotation to support the type check required for a string variable which should not contain a null value and to void the triggering of NullPointerException:

@NotNull String name;

A module can be designed to achieve this kind of type checking framework. When code is compiled including the NotNull module, the NotNull module checks the program for the potential problem and provides warnings if it detects any problem. Multiple type checking modules can be used for different type of errors. With the judicious use of type annotations and the presence of pluggable type checkers, you can write code that is stronger and less prone to error.

Annotations can have elements which can be named or unmade and these elements can be given values. In the Figure 12 annotations are being used to describing the properties of the node object of the class Node_Info:

```
41      @Documented
42      @Target(ElementType.CONSTRUCTOR)
43      @Retention(RetentionPolicy.RUNTIME)
44      @interface Node_information {
45
46          String node_name();
47
48          String Bs_alias();
49
50          String[] revokeList();
51      }
52      class Node_Info {
53
54          @NotNull
55          int node_id;
56          @Node_information(node_name = "Node_Read",
57                  Bs_alias = "Select_Customers",
58                  revokeList = {"Update", "Delete"})
59          Node_Info() {
60              node_id = 23;
61          }
62
63      }
```

Figure 12 Annotation Type Definition and Annotation Elements

In the Figure 12 we can see the use of Annotations in the expressions of node_id declaration, the @interface being used to define custom annotation and how this custom annotation provides information about the node. @Target tells that this annotation can be applied to a constructor; @Retention indicates how long annotations with the annotated type are to be retained.

3.6 State Diagrams for complex system representation

State diagrams are used to represent the states of the system when it is executing the programs to achieve a particular task in hand. They give an abstract description of the behaviour of the system. David Harel gave the model where the complex system can be explained using a diagrammatic approach representing the states of the system.

His work became the base of modelling the system using the UML and also designing models which can express the system in more efficient way. In Harel's models rectangles are used to denote the states and to represent the transition from one state to

another arrows are being used. The arrows are labelled to represent the event that caused the system to transit into another state(David HAREL, 1987). To understand Harel's model following example to the state diagram is given:

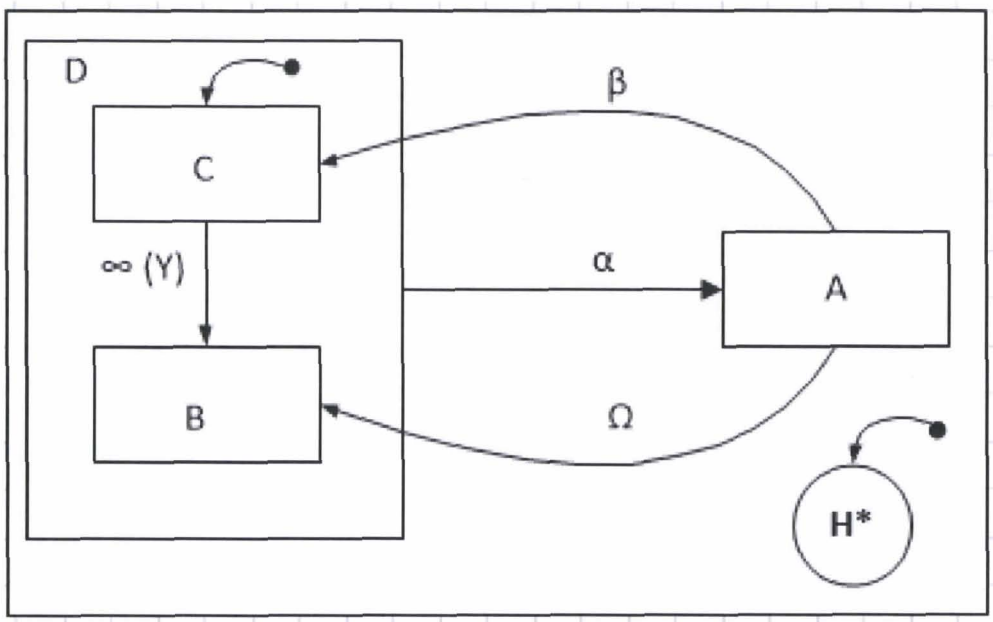

Figure 13 Example of Harel's state diagram

Above diagram shows the representation of system which enters into four stated namely A, B, C and D. The H^* state is the most recently visited history state. The arrows are labelled with the events which lead to the transition of the stated. By default the system enters into state C which is denoted by arrow with dot in the end. If the system doesn't have any history state then only it will enter into state C. State D comprises of state C and B, indicating if the system is either in state B or C then it is automatically entered into state D.

Harel's model is adopted in this work to represent the rules which a security expert need to keep in mind while designing the policies. These rules define the behaviour of the system while the execution of the predefined polcies.

One of the works which is influenced by Harel is by Il-Yeol Song et al(Il-Yeol Song, Ki Jung Lee, 2007); they published a paper where focus of their work was to represent how an object of a class makes transitions to different states during its execution life span by using the concept of specialization state. In their technique the meaningful states of the

class were identified before building the actual state diagram. Three identification rules were applied – state-valued attributes, association paths, and constraints.

4 Access Control Model Supporting Orchestration Of CRUD Expressions

An access control model is an abstraction of an access control mechanism which enforces access control policies specifying who can access what information under what circumstances.(Dae-Kyoo Kim, Pooja Mehta, Priya Gokhale, 2006). Defining an access control model requires the formulization of rules that can be applied while designing the access control policies, the scenarios where access control policies can be applied.

This chapter is divided into sections; section 4.1 explains the structure of access control policies, rules that should be followed while executing the access control policies are explained in sub section 4.1.1, the rules for designing policies are defined in sub section 4.1.2, the inter Graph execution is explained in sub section 4.1.3. Section 4.2 provides the proof of concept.

4.1 Structure Supported By the Presented Access Control Model

Presented model supports the design of the Access Control Polices in the form of graph structures. It allows mapping the implementation of the applications accessing the database into the design of the Access Control Policies. This section defines the rules which need to be followed while designing the Access Control policies.

In this thesis work design of the access control policies for validating the execution of authorized CRUD expressions for a particular user is implemented by considering the sequence of the CRUD expression as a graph structure connected by directed edges and regarded as a Policy Graph. Designing the access control policies in the form of graph structure allows the security experts to design the policies which can be applied in different scenarios.

This work inherits the concept of nodes and directed edges from the graph theory and applies this concept to the structure of the Access Control Policies. Therefore the structure of the access control policies consists of nodes which are connected via directed edges. Nodes represent the Business Schema which is a set of CRUD expressions that an application can execute and the relationship between the nodes is maintained by a directed edge.

Applying the graph structure for creating the policies requires the formulation of rules to be followed while designing the policy structure. The basic terminologies and rules associated with creation of access control policies are as follows:

- **Policy Graph**: directed graph structure whose nodes consists Business Schema. It is the access control policy which is used for validating the execution of CRUD expressions.
- **Terminating Node**: In a Policy Graph a terminating node's execution provides the user with two options, first if the terminating node has any outgoing edges then the user can still continue with the current Policy graph by requesting the Business schema of the target node, and second the user can request a new Policy graph and current Policy graph will no longer validates the execution of the CRUD expressions unless it is requested again.
- **Halt Node:** In a Policy Graph a node can request the execution another Policy Graph. This node is regarded as Halt Node.
- **Root Node**: it is the node of the graph where the execution of the Policy Graph starts by executing the Business Schema of the root.
- Policy graph must have at least one root and terminating node.
- In Policy graph the root and terminating nodes can be halt nodes.
- Policy graphs can have multiple root, terminating and halt nodes.
- The terminating node which has no outgoing edge cannot be a halt node.
- If a node has no outgoing edges then by default it is a terminating node
- If a node has a single outgoing edge and that edge is to itself then by default it is a terminating node
- If a node has no incoming edges then by default it is a root node.
- If node has only one incoming edge and that edge is a from the node itself then the node by default becomes a root node.

Presented model validates the execution of the sequence of CRUD expressions via a Policy Graph. Therefore if the execution of any CRUD expression belonging to se-

quence fails then the decision of rolling back the execution of sequence of CRUD expression is left to the application itself.

4.1.1 *Rules for the execution of the Policy Graph*

This section explains the rules for the execution of the policy graph. Policy graph validates client's execution of the CRUD expression. The rules for executing the Policy graph are:

- In Policy graph the execution of the graph always starts at the root node.
- The execution of the CRUD expression belonging to the Business Schema present on the source of the directed edge and the direction of the edge decides the next Business Schema to be executed.
- If the Policy graph contains single node than this node is a root and terminating node.

4.1.2 *Rules for designing the Policy Graph*

Following rules tries to map the scenarios of real world database driven application where user executes CRUD expressions to achieve a particular task.

4.1.2.1 *A single node can represent a complete Policy Graph*

A Policy Graph can be composed of a single Business Schema which allows the security manager to design the policy allowing the user to execute the CRUD expressions of one and only one Business Schema. Therefore a system can have one and only one state. Figure 14 shows an example of executing an update query for assigning the department to an employee. Figure 15 shows the structure of the Policy Graph for validating the execution of the single CRUD expression represented as a SQL *update* query.

```
public void AssignDepartmentToEmployee(int Emp_ID, String Emp_Department)
        throws SQLException {
    PreparedStatement ps = conn.prepareStatement("Update Employee_Information"
            + " set Emp_Department='@dept' where Emp_ID=@id");
    ps.setInt(2, Emp_ID);
    ps.setString(1, Emp_Department);
    ps.executeUpdate();
}
```

Figure 14 Java Code for execution of single CRUD Expression

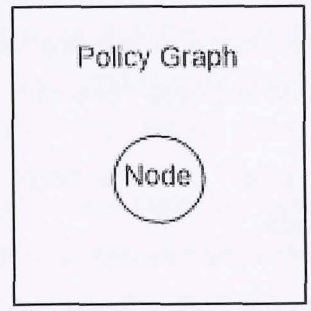

Figure 15 Policy Graph consisting of single node representing a State of the system

4.1.2.2 Policy graph can have a node which has a loop to itself

This access control rule is designed for scenarios where the execution of Business Schema is required in a loop. In this case the Update CRUD expression is called in the loop to assign the departments. This design of the access control policies addresses the scenario where the system remains in the same state until it completes required task as shown in Figure 17. Figure 16 shows a piece of java code where AssignDepartementEmployees() assigns the updates the table of information

The loop structure is not an endless loop, as the node is defined as the terminating node of the Policy graph. In applications where the system remains in a particular state by executing the CRUD expression in a loop, restriction for not executing other CRUDs can imposed by using this structure. This rule restricts the system to loop through a state

```
public void AssignDepartementEmployees(HashMap<Integer, String> EmpDepar)
        throws SQLException {
    {
        Set set = EmpDepar.keySet();
        Iterator it = set.iterator();
        while (it.hasNext()) {
            PreparedStatement ps = conn.prepareStatement
                    ("Update Employee_Information"
                + " set Emp_Department='@dept' where Emp_ID=@id");
            int EmpID = (int) it.next();
            ps.setInt(2, EmpID);
            ps.setString(1, EmpDepar.get(EmpID));
            ps.executeUpdate();
        }
    }
}
```

Figure 16 Java code for the execution of the CRUD expression in a loop

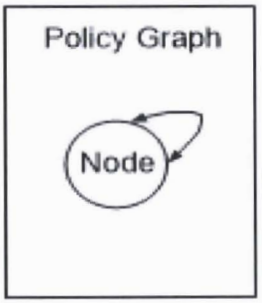

Figure 17 Policy Graph consisting of a node that has a loop to itself

4.1.2.3 Policy Graph can have multiple terminating nodes

A Policy Graph can have multiple terminating nodes. Figure 19 show the example where system can enter into one of the multiple states as the result of the execution one state, and these all can be regarded as terminating nodes.

4.1.2.4 Moving from single node to multiple nodes is supported

Figure 18 shows an example of Switch case statement where according to the age of the employee one of the various CRUD expressions is executed. This rule is designed

for scenarios where the execution of a CRUD expression can lead the system to one of the many possible states. Figure 19 shows the structure of the Policy graph for such scenarios.

In the example shown in Figure 18 the *select* query for choosing the Emp_ID and Emp_Age is a state which leads the system into one of the many states. Therefore the system presents in *switch* state enters into one of the many *case* statements state and performs the respected CRUD expression.

```
public void AnalyzeTablesAccordingAge() throws SQLException {
    PreparedStatement ps;
    ps = conn.prepareStatement("Select Emp_ID,Emp_Age from Employee_Information");
    ResultSet rs = ps.executeQuery();
    while(rs.next())
    {   int Emp_Age=rs.getInt("Emp_Age");
        int Emp_ID=rs.getInt("Emp_ID");
        switch (Emp_Age) {
            case 55:    ps=conn.prepareStatement("Update Employee_Information "
                        + "set Salary=2000");
                break;
            case 60:    ps=conn.prepareStatement("Delete from Working_Employees "
                        + "where Emp_ID=@id");
                ps.setInt(1, Emp_ID);
                break;
            case 30:     ps=conn.prepareStatement("Insert into Trainee"
                        + "(Emp_ID)values(@id)");
                ps.setInt(1, Emp_ID);
               break;
            default:
                ps=conn.prepareStatement("Insert into RetiredEmployee(Emp_ID)");
                ps.setInt(1, Emp_ID);
                break;
        }
     ps.executeUpdate();
```

Figure 18 Java code for Figure 19

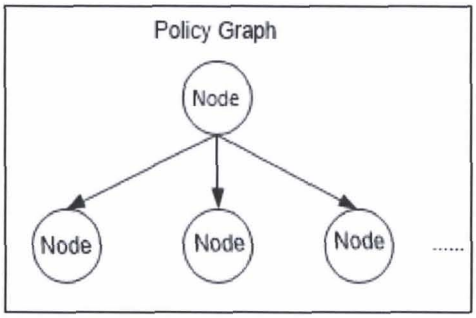

Figure 19 One state of the system leading to multiple states

4.1.2.5 Moving from multiple nodes to a single node is supported

Figure 21 shows an example written in java explaining a scenario where multiple *if, else if* always ends up with a *select* query. In this example execution of one of the if-else blocks will result into the execution of the one CRUD expression.

This rule allows the security experts to design the policies where the execution of one of the multiple CRUD expression results into the execution of one and only one CRUD expressions. This rule addresses the scenarios where multiple states of the system can lead to one of the one possible state as shown in Figure 21.

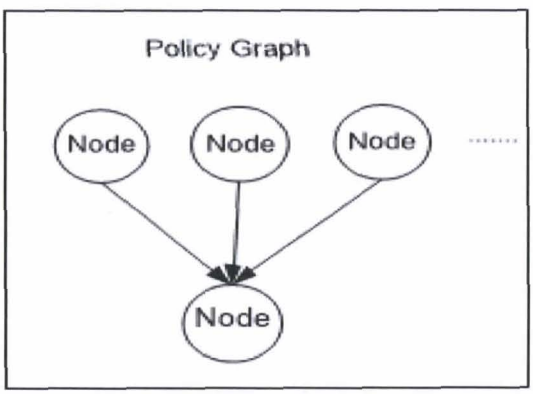

Figure 20 Multiple state of system leading to one state

```
public void PerformOperation(String query, HashMap<String, String> parameters)
        throws SQLException {
    PreparedStatement ps = null;
    Set set = parameters.keySet();
    Iterator it = set.iterator();
    if (query.equals("Insert")) {
        ps = conn.prepareStatement("Insert into Employee_Information(Emp_Name,"
                + ",Emp_Address)Values('@name','@address'");
        ps.setString(1, parameters.get(it.next()));
        ps.setString(2, parameters.get(it.next()));
    } else if (query.equals("Update")) {
        ps = conn.prepareStatement("Update Employee_Information set Emp_Address='@address'");
        ps.setString(1, parameters.get(it.next()));
    } else if (query.equals("Delete")) {
        ps=conn.prepareStatement("Delete * from Employee_Information where Emp_ID=@id");
        ps.setString(1,(parameters.get(it.next())));
    }
    ps.executeUpdate();
    ps = conn.prepareStatement("Select Emp_Name from Employee_Information");
    ResultSet rs = ps.executeQuery();
    while (rs.next()) {
        System.out.println("Emp_Name=" + rs.getString("Emp_Name"));
    }
}
```

Figure 21 One of the Multiple CRUD expression leads to single CRUD Expression

4.1.2.6 Policy graph restricted to single direction

Some scenarios require more rigid policies where the client can only follow a certain path, for example trying to withdraw money from an account. This policy allows the system to enter into one and only one state as the result of execution of CRUD expression.

Figure 22 shows java code to update the salary of an employee. In this example the execution of the update query depends on the result of the select query. The scenarios where the results of the execution of CRUD expressions of a state are used for the execution of CRUD expressions of the resulting state, access control policies must be designed to control order of the execution of state. Figure 23 shows the structure of the policy that can be designed to meet the scenario

```
public void UpdateSalary(int Emp_ID)  throws SQLException {
    PreparedStatement ps;
    ps = conn.prepareStatement("Select Salary from "
            + "Employee_Salary where Emp_ID=@id");
    ps.setInt(1, Emp_ID);
    ResultSet rs = ps.executeQuery();
    int salary = rs.getInt("Salary");
    if (salary > 1000) {
        salary = salary + 300;
        ps = conn.prepareStatement("Update Employee_Salary "
                + "set Salary=@salary where Emp_ID=@id ");
        ps.setInt(1, salary);
        ps.setInt(2, Emp_ID);
        ps.executeUpdate();
    }

}
```

Figure 22 Moving from one state to another in Single Direction

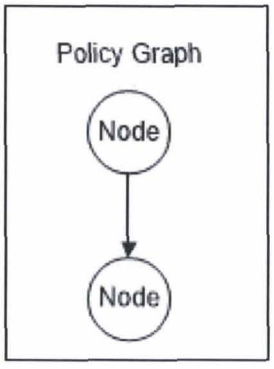

Figure 23 Execution following a single direction

By applying the above mentioned rules to the design of the access control policies security experts can design the policies to map the execution of the database driven applications to the structure of the access control policies.

4.1.3 *Inter Graph Execution*

This model supports the feature of halting the execution of an ongoing Policy Graph and starting a new authorized Policy Graph, and this is regarded as Inter graph communication. Popular programming languages supports method calling, which makes the piece of code inside that method reusable. Current Model uses this concept and supports the concept of reusing the policy graphs.

In Policy graphs The Policy graph whose execution is halted is regarded as Parent_Graph and the one whose execution is started as the result of halting is called Child_Graph. The rules that should be followed while implementing the Inter Graph communication are:

- Child_Graph's execution starts at the root node
- Only a halt node can call the Policy graphs which it is authorised to call.
- After the successful execution of the terminating node of the Child_Graph, system enters into the state in which the Parent_Graph was halted and based on the outgoing edges of the halt node the decision of the next Business Schema to be executed is made.

Figure 24 shows the block diagram where the Policy graph that has the halt node can request one of the many policy graphs that the halt node has authorization to call. Designing a graph where halt nodes are supported is a powerful feature as it allows the security experts to use the policies in different scenarios. But the security expert must take into consideration while designing the policy that by providing a halt node he gives the user a freedom to execute a different graph without finishing the current Policy graph. The security experts should consider before providing a privilege to the design.

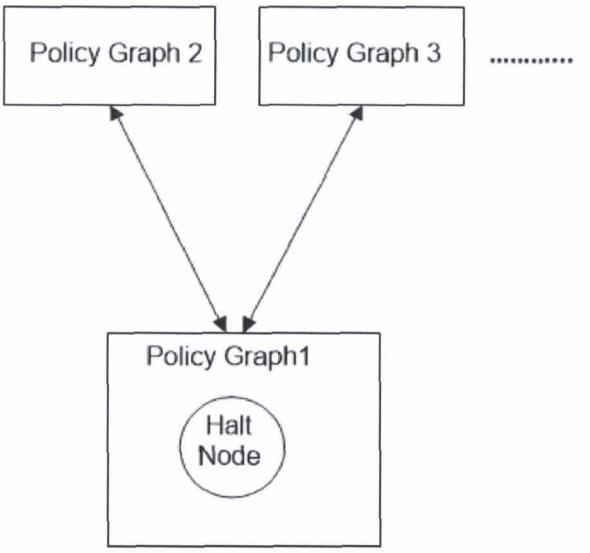

Figure 24 Inter- Graph execution Block diagram

4.1.3.1 State Diagram explaining the Inter-Graph Execution

For better understanding of the application of the rules a state diagram is presented in Figure 25. The state diagram representation used for representing the inter graph execution is based on Harel's model(David HAREL, 1987) .The diagram shown in Figure 25 contains six states named as A, B, C, Y, Z, H^*. 'TN' stands for the terminating node in the figure.

A is the default state for Policy Graph1 and B is the default state for Policy Graph2. The state H^* represents the most recent visited state of the system. Root node is marked with an incoming arrow carrying a dot which represents the default state of the systems. Whenever a CRUD expression of a Business Schema is executed the system enters into a new state. Returning back to the Home_Graph depends on the most recent visited state of the system. Suppose Policy Graph2 was visited from state Y, then most recent visited state of Policy Graph1 will be Y and the decision of the next Business schema is made based on the state the system is in. Numbering in the Figure 25 indicates a possible path that is being used for validating user's actions.

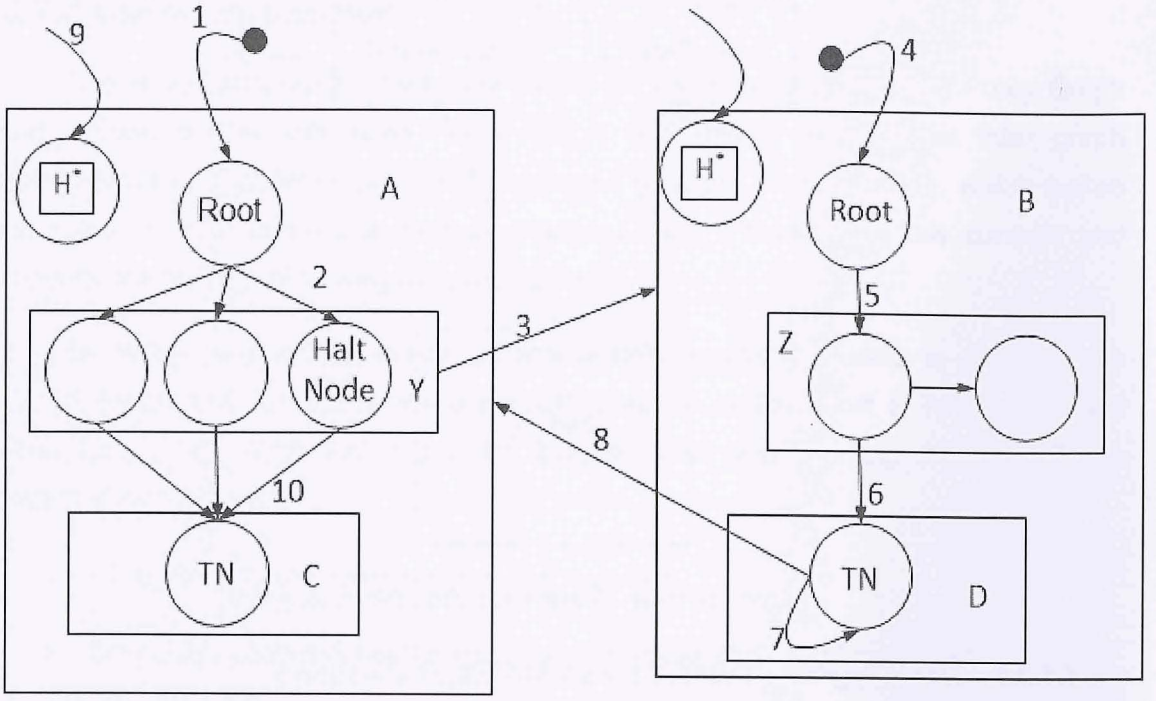

Figure 25 : state diagram representing the application of predefined policies

4.2 *Proof of concept*

This section provides a proof of concept for the Access Control Model defined by this thesis work. To implement the presented access control model S-DRACA is chosen as the base of this work and RBAC models has been chosen as the reference model for authenticating the clients. This section will bring light on the implementation of the Orchestration of CRUD expressions, configuration of the Policy graphs and their utilization for validating the actions of the user. This section also explains the modular structure which is inherited from S-DRACA, the changes made to the inherited components for supporting Policy graphs and the implementation of the presented Access Control Model using S-DRACA and RBAC.

4.2.1 *Block Diagram of the Access Control Model*

This section describes the behaviour of the various components that are inherited from S-DRACA and the changes that are made to these components for supporting the presented Access Control Model.

Current model inherits the modular structure of the S-DRACA but changes the implementation of some of the components to support the access control policies which are now designed as graph structures. S-DRACA provided the security layer which provided the features of authentication and encryptions. In this work the security layer which is provided by S-DRACA is not altered by any means.

The problem address with this work is freedom that a user can have after he is authenticated to access a resource. To provide a solution to the problem RBAC model has been chosen as a reference model. Figure 26 shows the block diagram where the different components that are inherited from S-DRACA are shown.

Access control policies are stored in the Policy Server which is a relational database and holds the access control policies in the form of relational tables. The client connects to the Policy Manager and he is authenticated by the access control model that is deployed on the server side. In current implementation the client is authenticated on the bases of RBAC model. After the authentication of the client, Policy Manager gets the authorized policies which are stored in the policy server and generates the Policy graphs. The Policy graph is then sent to the Policy extractor, and Policy extractor generates the interfaces for the Business Schemas.

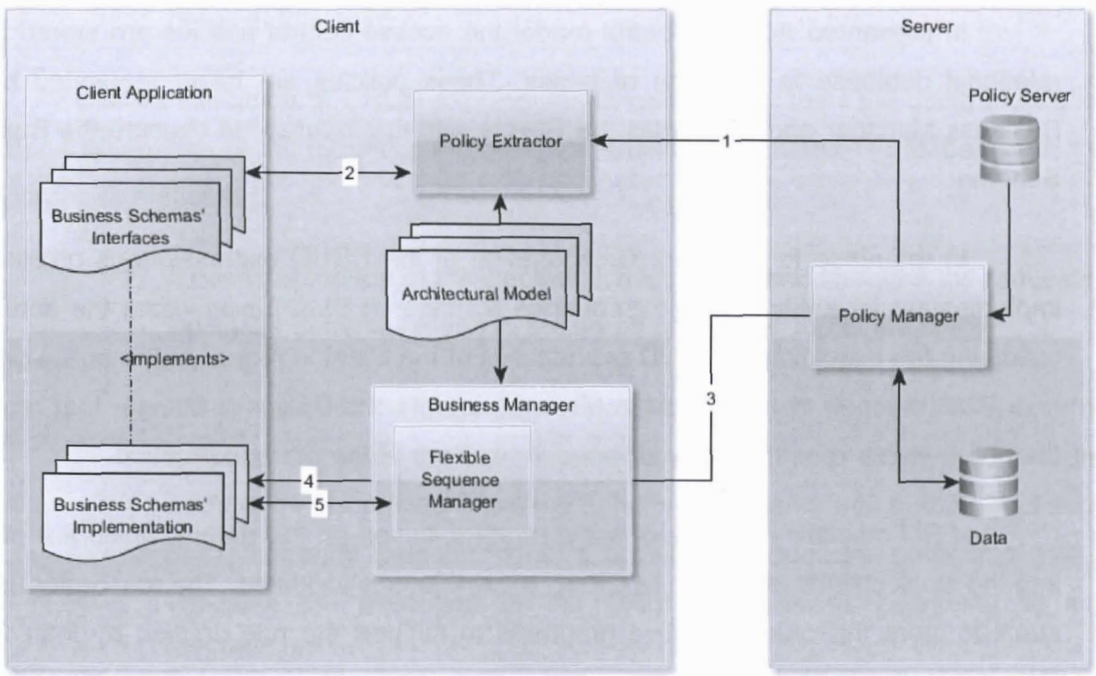

Figure 26 Block Diagram representing the work flow of Presented work

At run time the Business Manager request the Policy graphs from the Policy Manager and also implements the interfaces generated by the Policy extractor. The client requests the next Business schema to be executed from the Business Manager. If the client tries to execute a Business Schema which is not allowed to be executed then Business Manger throws an exception based on the policy graph. The Policy manager relays the communication of the Business Manager and the Policy server.

In current implementation Business Manger holds another component Flexible Sequence Manger. This component allows the inter-graph execution and the validation of the client operations in regards to the Policy graph it has. The implementation of the modified components is shown in section 4.2.3.

4.2.2 CRUD Orchestration

The word Orchestrate means organizing or arranging something via a planned and clever way. Orchestration of the CRUD expression in presented work means arranging the CRUD expression in the form of directed graph that can be used by an entity on the client side for providing the service of validating the execution of the CRUD expressions of the client and generating the interfaces for the execution of the authorized CRUD expressions.

In presented Access Control model the access control policies are stored in the relational database in the form of tables. These policies are being requested by the Business Manager and it provides the Clients with the interface to execute the Business Schema.

In reference to this work Orchestration of the CRUD expressions is obtained by implementing an entity "Flexible Sequence Manager (FSM)" for providing the service of validating the execution of CRUD expressions of the client in regard to the policy graph it has. FSM is a part of the Business Manager and it's the Business Manger that requests the policy graph from the Policy Manager on behave of the client application.

FSM maintains a stack of active graphs, based on the graphs which are pushed into the stack client's execution of CRUD expressions is validated. The reason for using a stack to store the order of active graphs is to support the rule defined by Inter-Graph Communication in section 4.1.3.

Whenever client application requests the execution of a new policy graph, FSM performs the following operation:

1. It pops the policy graph which is on top of the stack of active graphs.
2. Checks if the current node that is executed to validate the execution of CRUD expression is a halt or terminating node.
3. If it is a halt node then it checks if the requested Policy graph can be executed from this node or not. If both conditions holds true then it pushes back the current Policy graph into the stack of active graphs, push the new Policy graph into the stack and start the execution of the requested new Policy graph by executing the root node.
4. If it is a terminating node with outgoing edges then it push back the current Policy graph into the stack, push the new Policy graph into the stack and start the execution of the requested new Policy graph by executing the root node.
5. If it is not a halt or terminating node then it throws an exception for the client, as he is not authorized to request the execution of the new Policy graph at this point.

Client's execution of CRUD expressions is validated by the FSM using the Policy graph. When the client requests for the next Business Schema, FSM performs the following operations:

1. It peeks at the top of the stack of active graphs and checks if the requested Business Schema belong to the graph present on top of the stack
2. If it is a part of the graph present on top of the stack, then FSM pops the graph and check if the currently executed node has a directed edge to the node of the requested Business Schema and the source of the direct edge is the currently executed node. If both of the conditions holds true then it validates the execution of the CRUD expressions belonging to that schema.
3. If it is not the part of the graph present on top of stack of active graphs then it performs the check for the execution of the new Policy graph.

The implementation of the stack of active graphs is shown in section .

FSM acts like a central repository which authenticates the execution of the next authorized business schema. Figure 27 shows the entire structure for the CRUD orchestration:

1. Retrieving stored policies from the Policy server
2. Sending policies as objects containing the Policy graphs for authorized user
3. Authenticating next CRUD expressions to be executed
4. Result of CRUD execution from the client to the Policy manager and response of the Policy Manager.

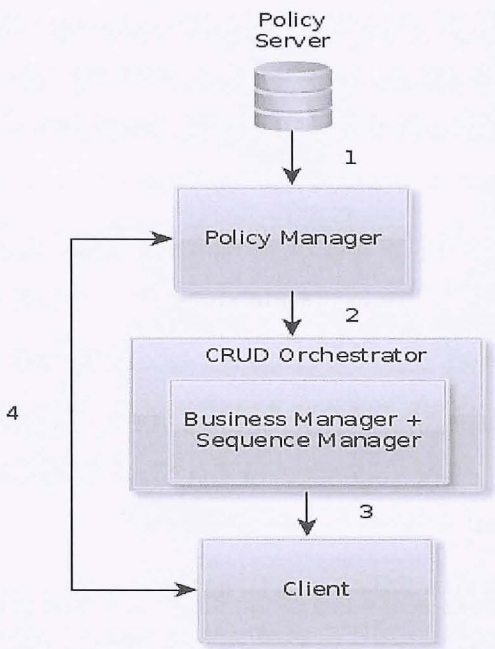

Figure 27 Orchestration of CRUD expressions

4.2.3 Implementation of the Access Control Model using S-DRACA and RBAC

This section explains the implementation of the access control policies, how they are being sent to the Business Manager and Policy Extractor, the generation of the interfaces using annotation and reflections, how Business Manager provides the validation of CRUD expression and provide the next Business Schema interface to the client for execution of CRUD expressions.

4.2.3.1 *Implementing the Access Control Policies*

Design of the security policies to control the execution of CRUD expressions allocated for the role is done by a security manager who considers the fact that RBAC model is chosen as a reference model. These policies are stored in the databases and when a client connects to the server, these policies are being sent by the Policy Manager to the Business Manager and Policy Extractor.

Providing flexibility for designing these policies was one of the most challenging tasks faced in the completion of this work. In database applications users are not restricted to execute a uni-directed sequence of business schemas but it's the execution of the business schema that takes the system to a state where user can execute one of many business schemas or execute a single schema. Figure 28 shows the entity relationship diagram describing how the policies are stored in the database.

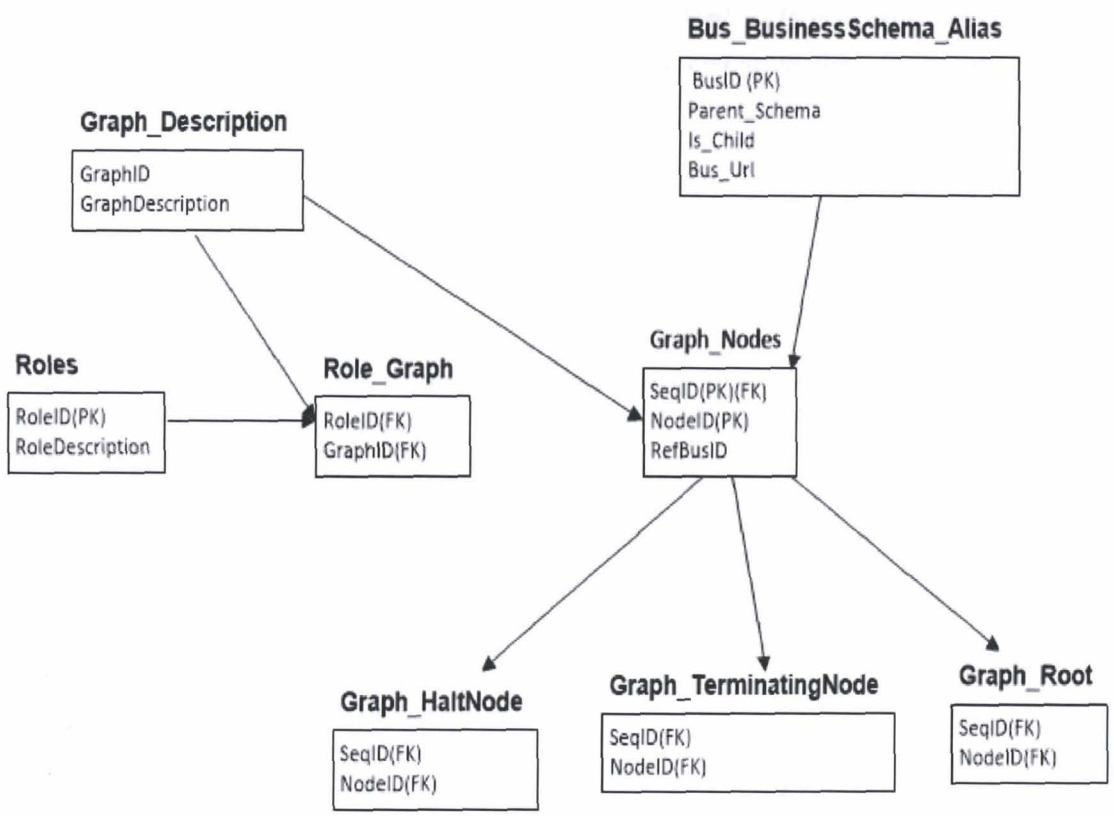

Figure 28 Entity Relationship Diagram for different tables for the Policy Graph

For current work policies are stored in Policy Server which is a relational database. Policy Configurator is a module inherited from S-DRACA which acts as a security expert

for designing the access control policies and storing them into the database. It inserts the policies into the database by using SQL queries. Figure 31 shows an SQL table for storing the nodes of the different graphs that are present in table shown by Figure 29.

	GraphID	Description
1	1	Graph1
2	2	Graph2
3	3	Graph3
4	4	Graph4

Figure 29 SQL table for storing the Policy Graph description

Figure 30 shows the SQL queries for inserting the names and the nodes that a policy graph have into the table. After the values are inserted into these tables, Policy Configurator needs to formulate the design of the access control policy by inserting the edge information of the nodes of the graphs, specifying the root nodes/node of the graph, specifying the terminating node/nodes of the graph and providing the information about the halt node/nodes.

```
pstmt = conn.prepareStatement(
        "INSERT INTO PolicyGraphs (GraphID, Description) "
        + "VALUES (4, 'Graph4')");
pstmt.executeUpdate();

pstmt = conn.prepareStatement(
        "INSERT INTO BS_GraphNodes (SeqID, NodeID, RefBusID) "
        + "VALUES (1, 1, (SELECT Bus_id FROM Bus_BusinessSchema "
                + "WHERE Bus_url = 'S_Customers.IS_Customers'))");
pstmt.executeUpdate();
```

Figure 30 SQL queries to insert values in the tables shown in Figure 31 and Figure 29

	SeqID	NodeID	RefBusID
1	1	1	4
2	1	2	1
3	1	3	2
4	2	1	4
5	2	2	3
6	3	1	4
7	3	2	1
8	3	3	2
9	3	4	3
10	4	1	4
11	4	2	1
12	4	3	2
13	4	4	3

Figure 31 : SQL table for all the nodes of the Graphs of Figure 29

The connection between the nodes of the graph is shown by the table Figure 32. This table provides the information of the directed edge by regarding the nodes as parent node and child node. The source of the edge is the parent and target is the child.

	SeqID	Parent_Node	Child_Node
1	1	1	1
2	1	1	2
3	1	2	3
4	2	1	2
5	3	1	2
6	3	1	3
7	3	1	4
8	4	1	4
9	4	2	4
10	4	3	4

Figure 32 SQL table specify the parent and child nodes of different graphs

As there can be more than one halt, terminating and root nodes, therefore three different tables are created in which the respected information is provided. Figure 33 shows the table where the root node of the various graphs is inserted.

	SeqID	NodeID
1	1	1
2	2	1
3	3	1
4	4	1
5	4	2
6	4	3

Figure 33 SQL table for the root nodes of different graphs

	SeqID	NodeID	AllowedChildGraphID
1	1	2	2

Figure 34 SQL table for halt nodes

Figure 34 shows an SQL table for the halt nodes. The column AllowedChildGraphID specifies which graphs can be requested from this node. To implement Inter-Graph Execution the permission of requesting the new Policy graph is implemented on node level rather than Graph level. The reason for doing this is to provide the security expert with the tool that he can design the policies where the user can only start inter-graph execution on specific nodes. If the permission to start the inter-graph execution is provided at graph level then user can be at any node in the graph and can request the new policy graph, and this can be avoided if needed.

The terminating nodes information is given by the table shown in Figure 35. Terminating nodes indicates that the execution of the Policy Graph has ended based on the policies defined in section 4.1

	SeqID	NodeID
1	1	3
2	2	2
3	3	4

Figure 35 SQL table for terminating nodes

To implement these policies, graphs are considered to be the basic structure. Authorizing the path that a user can follow to accomplish a task is done by creating the directed graph structures where the nodes carry the following information:

- The business schema that can be executed on that node
- ID of the node in that particular graph

- If the node is a terminating node or not
- If the node is a root node or not
- The IDs of the other Policy graphs whose execution can be requested after the execution of the Business Schema of this node (if it is a halt node).
- The list of the Business Schemas that are revoked for this node
- List of IDs of the nodes that are child of this node (edges)

Figure 36 shows the structure of the class Node_Info. The nodes of the Policy graph are of type Node_Info. Policy manager gets the policies from the Policy server and then creates directed graphs which are regarded as Policy graphs.

```
public class NodeInfo {
    public Integer nodeID;
    public String BSurl;
    public List<Integer> edges = new ArrayList<>();
    public List<String> revokeList = new ArrayList<>();
    public boolean isRootNode = false;
    public List<String> haltNodeCallable = new ArrayList<>();
    public boolean isTerminatingNode = false;
```

Figure 36 Java code implementing the class Node_Info

Policy manager is the only entity in the project that communicates directly to the Policy Server, the requests from the Business Manager and Policy Extractor for getting the stored access control policies are handled by Policy Manager. The result of the execution of the Business Schema from the clients is mediated by the Policy Manager to the underlying database.

As the policies are now graph structure therefore the Policy Manager is modified to support the change in structure of the policies. Policy Manager gets the policies which are stored in the Policy Server by using the SQL queries. Policy Manager creates the graphs that are authorized for a particular role.

After this information is retrieved from the database the Policy Manager creates the Policy graph. Figure 37 shows the java code which for creating the structure of the Policy Graph. Nodes of the graph are of type Node_Info which is a class shown in Figure 36.

```
    PreparedStatement pstmt = conn.prepareStatement
(Select_Sequences_ID_Position_BusUrl_By_RoleReference);
    pstmt.setString(1, role);
    ResultSet rs = pstmt.executeQuery();
    while (rs.next()) {
        int seq = rs.getInt(1);
        int nodeN = rs.getInt(2);
        String beurl = rs.getString(3);
        pstmt = conn.prepareStatement(Select_Edges_Of_NodeID);
        pstmt.setInt(1, seq);
        pstmt.setInt(2, nodeN);
        ResultSet edgesRS = pstmt.executeQuery();
        List<String> revokelist = getRevokeList(seq, nodeN);
        if (!sequences.containsKey(seq)) {
            sequences.put(seq, new HashMap<>());        }
        Map<Integer, NodeInfo> seqdata = sequences.get(seq);
        ArrayList<Integer> edges = new ArrayList<>();
        while (edgesRS.next()) {
            edges.add(edgesRS.getInt(1));
        }
        NodeInfo ni = new NodeInfo(nodeN, beurl, edges, revokelist);
       ni.isRootNode = isRoot(seq, nodeN);
        ni.haltNodeCallable= getNodeCallable(seq,nodeN);
       ni.isTerminatingNode=isTerminatingNode(seq,nodeN);
       seqdata.put(nodeN, ni);
```

Figure 37 Java code for getting the graphs for each Role

At compile time Policy Extractor requests for the policies from the Policy Manager and Policy Manager responds to the request of the Policy Extractor by writing the structure of the Policy Graph using PrintWriter. Figure 38 shows java code, where the Policy Manager writes the information regarding the Policy Graph using PrintWriter object.

```
Map<Integer, Map<Integer, NodeInfo>> sequences =
        dbHandler.getSequenceInfoForRole(inputFields[1]);
for (int seqID : sequences.keySet()) {
    Map<Integer, NodeInfo> sequence = sequences.get(seqID);
    for (int nodeID : sequence.keySet()) {
        System.out.println(String.format("node %d %d %s",
                seqID, nodeID, sequence.get(nodeID).BSurl));
        out.println(String.format("%d %d %s",
                seqID, nodeID, sequence.get(nodeID).BSurl));
        out.flush();
```

Figure 38 Java code of Policy Manger's reply to Policy Configurator

4.2.3.2 Extraction and Application of Access Control Policies

Policy Extraction:

Policy Extractor sends message to Policy Manager requesting the list of access control policies and according to these policies interfaces are generated at compile time by the policy annotations. Java reflection and annotations are being used to generate the interfaces. This part is inherited from S-DRACA.

As policies are designed in the form of graph structures, the Policy Extractor is now modified to support these structures. It sends a request to the Policy manager requesting the information about the policies and receives the policies in the form of strings as shown in Figure 38.

As Policy Extractor get the reply from the Policy Manger, it starts to reconstruct the Policy graph using the replies. A custom message protocol ensures that the data sent is correctly interpreted on the Policy Extractor sides, and the same adheres on the Policy Manager side. Figure 38 shows the java code for the creating the Policy graph from the replies of the Policy Manager.

After the policy graph is reconstructed the interfaces for the Business Manager can now be generated by using the java annotation processor and the java reflection. Figure 40 shows the method which is being used to generate the interfaces for the Business Manager. The StringBuilder object is being used to construct the interface by appending the information about the business schema of the node, its child nodes etc.

```
out.println("getSeqInfo " + role);
out.flush();
String line;
while (!(line = in.readLine()).equalsIgnoreCase("END")) {
    String[] resp = line.split("[ ]");
    if(resp.length >= 4) {
        Integer seq = Integer.parseInt(resp[0]);
        Integer nodeID = Integer.parseInt(resp[1]);
        Integer edge = Integer.parseInt(resp[2]);
        if (!orqinfo.containsKey(seq)) {
            orqinfo.put(seq, new HashMap<>());
        }
        Map<Integer, NodeInfo> seqdata = orqinfo.get(seq);
        if(!seqdata.containsKey(nodeID)) {
            NodeInfo nodeInfo = new NodeInfo(nodeID);
            nodeInfo.BSurl = "BSerrorURL";
            nodeInfo.edges.add(edge);
            seqdata.put(nodeID, nodeInfo);
        } else {
            seqdata.get(nodeID).edges.add(edge);
        }
```

Figure 39 Java code to create Policy Graph from the Policy Manager's reply

```
private static void generateNextMethods(StringBuilder strb, Class c,
        Map<Integer, Map<Integer, NodeInfo>> orqinfo) {
    String beurl = c.getPackage().getName() + "." + c.getSimpleName();
    for (Integer seq : orqinfo.keySet()) {
        for (Integer nodeID : orqinfo.get(seq).keySet()) {
            String gotbeurl = orqinfo.get(seq).get(nodeID).BSurl;
            if (gotbeurl.equalsIgnoreCase(beurl)) {
                strb.append("// : " + orqinfo.get(seq).get(nodeID) + "\n");
                for (Integer edgeID : orqinfo.get(seq).get(nodeID).edges) {
                    strb.append("    public ");
                    strb.append(orqinfo.get(seq).get(edgeID).BSurl).append(" ");
                    strb.append("nextBE_G").append(seq).append("_edge").append(edgeID);
                    strb.append("(int crud, ISession session) throws LocalTools.BTC_Exception;\n");
```

Figure 40 Method to generate Interfaces for Business Manager

Policy Application

Business Manager implements the interfaces which are generated by the Policy Interface generator and provides the client application with the interfaces where the client can execute the authorized Business Schemas. The Business Managers holds another module knows as FSM, it is being used to enforce the rules of the presented access control model.

Business Manager has been modified to support the validation based on a Policy graph. Business Manager's module FSM performs the validation of the CRUD expression that the client is trying to execute by requesting a Business Schema. If the client is not allowed to the CRUD expression it throws an exception. It's the FSM that enforces the rules defined by the presented Access Control Model.

```
public void updateAliveBEs(Map<Integer, List<NodeInfo>> runningSeqs, String beUrl) {
    for (Integer seq : runningSeqs.keySet()) {
        if (!runningSeqs.get(seq).get(lastExecutedNode).isEndNode()) {
            for (Integer nextEdge : runningSeqs.get(seq).get(lastExecutedNode).edges) {
                String edgeBS = getBus(runningSeqs.get(seq).get(nextEdge));
                setAuthorizedBS(edgeBS);
                List<String> values = getRevokeList(runningSeqs.get(seq).get(nextEdge));
                if (edgeBS.equals(beUrl)) {
                    besalive.removeAll(values);
                }
            }
        }
    }

    besalive.add(beUrl);
}
```

Figure 41 Java code for the management of Validation of next Business Schema

Figure 41 shows the function which is called when the user request for the next Business Schema. This function checks if the last executed node is not an End Node (a node having no outgoing edges or just a loop to itself) then it gets its outgoing nodes and puts the target nodes Business Schemas into the list of alive Business schemas.

The validation of a Business Schema execution is done when the client executes the requested Business Schema. If a client is executing the business schema of a halt node then the callable sequence's root nodes' business schemas are inserted into the

branches of the active sequence along with the business schemas of the outgoing edges it has with the current executed node. If the next business schema executed by the client is the business schema of a callable sequence then this callable sequence is pushed onto the stack of active sequences. Following the node after the halt node of the same policy graph makes the previously added policy graphs to be removed from the current branches of the active sequence. In case of a terminating node, where it was requested a next business schema (in case there are any outgoing edges) then the branches of the active sequence is filled with the Business Schema of its target nodes.

5 Conclusion and Future directions

An attempt to bring flexibility in the designing of the security policies for controlling the execution of the CRUD expressions has been made. The introduction of the FSM allows providing the orchestration of the CRUD expressions. Security managers can design the policies that can be applied in many areas of business world based on the rules defined for the support of this model. The decision of formalizing the rules supported by this model is based on the real world needs and demands. This model can be used to track the path that a user follows as the FSM component knows what the client is executing. This feature can be applied to many scenarios such as to know on which level majority of users get stuck while playing a game and hence the analysts can use this information to know that where the problem is, in the design of the game.

Currently the exchange of information between the Policy Manager, Policy Extractor and Business Manager is done by writing the data onto the output and input streams. This information exchange can only be possible if there is an agreement to use a specific message format. In future work this can be modified by using objects for sending and receiving data.

The Security Configurator is a file which is manually written to enter the policies into the database. This component can be upgraded so that the security manager does not have to write the access control policies one by one. In current structure if need to modify a specific access control policy we need to run the Security Configurator all over againto change the policy in the database, this can be avoided.

6 Works Cited

David HAREL. (1987). Statecharts: A visual formalism for complex systems. Science of Computer Programming, 231 - 274.

Diogo José Domingues Regateiro. (2014, july). A secure, distributed and dynamic RBAC for relational applications. Aveiro, Portugal.

Edgar F.Codd. (1990). The Relational Model for Database Management version 2. Addison-Wesley Publishing Company.

Ravi Sandhu. (2004). A Perspective on Graphs and Access Control Models., (pp. 2–12). Berlin.

Ravi Sandhu. (2004). A Perspective on Graphs and Access Control Models. In R. Sandhu, Graph Transformations (pp. 2-12). Rome,Italy: Springer Berlin Heidelberg.

SQL Server Mangement Studio. Retrieved November 11, 2015, from Microsoft: msdn.microsoft.com/en-us/library/ms174173(v=sql.110).aspx

ABAC. (2015, May 6). ATTRIBUTE BASED ACCESS CONTROL (ABAC) - OVERVIEW. Retrieved November 17, 2015, from NIST: http://csrc.nist.gov/projects/abac/

Anthony J. Bonner. (1997). Transaction Datalog: a Compositional Language for Transaction Programming. Proceedings of the Sixth International Workshop on Database Programming Languages, (pp. 303-322). Colorado.

Ausanka-Crues, Ryan. Methods for Access Control:Advances and Limitations. Claremont, California.

Boniface Hicks, D. K. (2007). Jifclipse: Development Tools for Security-Typed Languages. ACM SIGPLAN Workshop on Programming Languages and Analysis for Security . San Diego, California: PLAS.

Brice Morin, Tejeddine Mouelhi,Franck Fleurey,Yves Le Traon,Olivier Barais and Jean-Marc Jézéquel. (2010,). Security-Driven Model-Based Dynamic Adaptation., (pp. 20–24). Belgium.

Caulfield, T.; Pym, D. (2015). Improving Security Policy Decisions with Models. Security & Privacy, IEEE (pp. 34 - 41). IEEE.

Chlipala, Adam. (2010). Ur: Statically-Typed Metaprogramming with Type-Level Record Computation. PLDI '10 Proceedings of the 31st ACM SIGPLAN Conference on Programming Language Design and Implementation (pp. 122-133). New York: ACM.

Chlipala, Adam. (2015). Ur/Web: A Simple Model for Programming the Web. POPL'15, 153-165.

D.Richard Kuhn, Ramaswamy Chandramouli,David F.Rerraiolo, Ravi Sandhu,Serban Gavrila. (2001). Propesed NIST Standards for Role Based Access Control. ACM Transactions On infroamtion and system security, 224-274.

Dae-Kyoo Kim, Pooja Mehta, Priya Gokhale. (2006). Describing Access Control Models as Design Patterns Using Roles. Proceedings of the 2006 conference on Pattern languages of programs.

David F. Ferraiolo and D. Richard Kuhn . (1992). Role-Based Access Controls. National Computer Security Conference. baltimore.

Il-Yeol Song, Ki Jung Lee. (2007). Developing State Diagrams Using a State Specialization Technique. 86–95.

Jeff Zarnett,Mahesh Tripunitara,Patrick Lam. Role-based access control (RBAC) in Java via proxy objects using annotations. Proceeding of the 15th ACM symposium on Access control models and technologies - SACMAT '10.

Jeffrey Fischer, Daniel Marino, Rupak Majumdar, and Todd Millstein. (2009). Fine-Grained Access Control with Object-Sensitive Roles.

Jgrapht. Retrieved november 12, 2015, from Jgrapht Organization: http://jgrapht.org/

Jif: Java + information flow. Retrieved November 22, 2015, from http://www.cs.cornell.edu/jif/

Kangsoo Jung, Seog Park. (2013). Context-Aware Role Based Access Control Using User Relationships. International Journal of Computer Theory and Engineering.

Lars E. Olson, Carl A. Gunter, and P. Madhusudan. (2008). A Formal Framework for Reflective Database Access Control Policies. ACM.

Manogna Thimma, Tsam Kai Tsui, Bo Luo. (2015). HyXAC: Hybrid XML Access Control Integrating View-Based and Query-Rewriting Approaches. IEEE Transactions on Knowledge and Data Engineering,, 2190 - 2202.

Netbeans IDE features. Retrieved November 12, 2015, from Netbeans Org: https://netbeans.org/features/index.html

Neward, Ted. IBMDeveloperWorks. Retrieved November 22, 2015, from IBM: http://www.ibm.com/developerworks/library/j-5things1/

Niklas Broberg, Bart van Delft,David Sands. (2013). Paragon for Practical Programming with Information-Flow Control. In B. v. Niklas Broberg, Programming Languages and Systems (pp. 217-232). Springer International Publishing.

Oracle Java Documentation. Retrieved November 12, 2015, from Oracle: https://docs.oracle.com/javase/tutorial/jndi/objects/serial.html

Oracle. Java Tuitorials. Retrieved November 22, 2015, from Oracle: https://docs.oracle.com/javase/tutorial/jndi/objects/serial.html

Óscar Mortágua Pereira, Diogo Domingues Regateiro, Rui L. Aguiar. (2014). Extending RBAC model to Control Sequence of CRUD Operations. Int. Conf. on Software Engineering and Knowledge Engineering. Vancouver.

Óscar Mortágua Pereira, Diogo Domingues Regateiro, Rui L. Aguiar. (2014). Role-Based Access Control Mechanisms Distributed, Statically Implemented and Driven by CRUD Expressions. IEEE International Symposium on Computers and Communications. Madeira.

Óscar Mortágua Pereira, Diogo Domingues Regateiro, Rui L. Aguiar. (2015). Secure, Dynamic and Distributed Access Control Stack for Database Applications. SEKE 2015 - Intl. Conference on Software Engineering and Knowledge Engineering. Pittsburgh.

Paragon. Retrieved November 2015, from http://www.cse.chalmers.se/research/group/paragon/

Pierangela Samarati, Sabrina de Capitani di Vimercati. (2001). Access Control:Policies,Models,Mechanisms., (pp. 137–196).

Ravi S.Sandhu, Edward J.Coyne, Hal L.Feinstein and Charles E.Youman. (1995). Role_Based Access Control Models.

Rodica Tirtea, Demosthenes Ikonomou,Slawomir Gorniak,Panagiotis Saragiotis. (2011). Survey of accountability, trust, consent, tracking, security and privacy mechanisms in online environments. European Union Agency for Network and Information Security.

ROGER NEEDHAM, RICK MAYBURY. (2008). Access Control. (pp. 93-128). Wiley.

S. Jha, O. Sheyner , J. Wing. (2002). Two Formal Analyses of Attack Graphs. Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE (pp. 49 - 63). IEEE.

S. Sumathi, S. Esakkirajan. (2007). Fundamentals of Relational Database Management System. New York: Springer Berlin Heidelberg.

S. Sumathi, S. Esakkirajan. (2007). Transaction. In S. E. S. Sumathi, Fundamentals of Relational Database Management Systems (pp. 319-326). Berlin: Springer-Verlag.

Samarati, Ravi S.Sandhu Pierangela. (1994). Access Control: Principle and Practice. Communication Magazine IEEE, 40-48.

The Java Tuitorials. Retrieved November 22, 2015, from Oracle: https://docs.oracle.com/javase/tutorial/java/annotations/index.html

Type annotations and Pluggable type Systems. Retrieved November 22, 2015, from Oracle: https://docs.oracle.com/javase/tutorial/java/annotations/type_annotations.html

Vincent C. Hu, David F. Ferraiolo , D. Rick Kuhn. (2006). Assessment of Access Control Systems. Gaithersburg,: National Institute of Standards and Technology.

Vincent C. Hu, David Ferraiolo,Rick Kuhn,Adam Schnitzer,Booz Allen Hamilton,Kenneth Sandlin,Robert Miller,Karen Scarfone. (2014). Guide to Attribute Based Access Control (ABAC) Definition and Considerations. NIST.

Virgil D. Gligor, Serban I. Gavrila,David Ferraiolo. (1998). On the Formal Definition of Separation-of-Duty Policies and their Composition. IEEE Symposium on Security and Privacy. Oakland, California.

Yanjie Zhou, Li Ma , Min Wen. (2015). A Multi-level Dynamic Access Control Model and Its Formalization. Information Science and Control Engineering (ICISCE), 2015 2nd International Conference (pp. 23 - 27). IEEE.

Estes anexos só estão disponíveis para consulta através do CD-ROM.
Queira por favor dirigir-se ao balcão de atendimento da Biblioteca.

Serviços de Biblioteca, Informação Documental e Museologia
Universidade de Aveiro