American Journal of Advanced Research, 2021, 5–1
July. 2021, pages 06-12
doi: 10.5281/zenodo.5112633
http://www.ibii-us.org/Journals/AJAR/
ISBN 572-8849 (Online), 2572-8830 (Print)

Database Web Application for Administering Spatio-Temporal Access Control Policies

Miguelangel Trevino[1], Mustafa Al Lail[2]

[2]Department of Engineering, Address 5201 University Boulevard, Lamar Bruni Vergara Science Center 312, Laredo, TX 78041-1900

[1]Email: miguelangeltrevino@dusty.tamiu.edu

[2]*Email: mustafa.allail@tamiu.edu

Received on June 06, 2021; revised on July 14, 2021; published on July 18, 2021

Abstract

Governmental and business organizations use the standard authorization model─ Role-based access control (RBAC) ─ to specify and administer access policies for electronic resources. In RBAC-based applications, access is granted or denied based on users' credentials. However, the RBAC model lacks features that allow applications to determine access based on time and location, spatio-temporal information. This access requirement is important for a growing number of mobile applications. Researchers have proposed new access control models to accommodate organizations' reliance on mobile applications. The General Spatio-temporal Role-Based Access Control model (GSTRBAC) is a model that incorporates time and location constraints as additional factors to grant access to resources. This paper presents the results of our undergraduate research project on creating a relational database that provides a way to store and retrieve GSTRBAC policy information. Further, the paper describes a web application that security analysts can use to administer GSTRBAC policies.

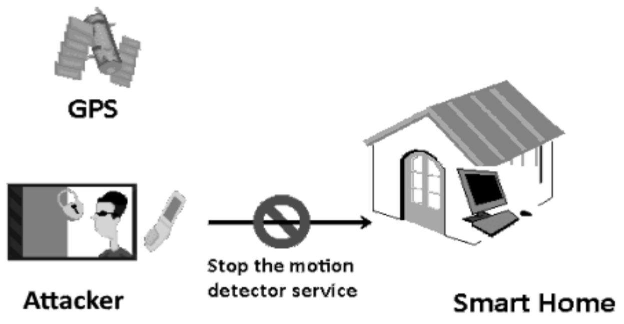
Keywords: Authorization, Role-Based Access Control, Mobile Application, Time, Location

1 Introduction

With the advent of technological advancement and wireless communication, more governmental and private sector organizations rely on mobile applications to support their beneficiaries. Such applications abide by the specialties of mobile technology attempting to gather information on users' environments. These environmental conditions, such as time and location, can be coupled to a user's credentials to allow for more advanced authorization. Figure 1 depicts an example of the use of location to allow access to smart home systems. Using the Global Positioning System (GPS), the location of the mobile device is determined to be outside the smart home, where access to the home's motion detector system should not be allowed. Such a policy, known as spatio-temporal policy, ensures the protection of homeowner's security in the event of a lost or stolen device.

The Role-based access control model (RBAC) is the standard access control model used by multiple organizations [1]. This model, however, is incapable of expressing time and location constraints. Thus, RBAC has been extended to accommodate spatio-temporal requirements [2]-[4]. The

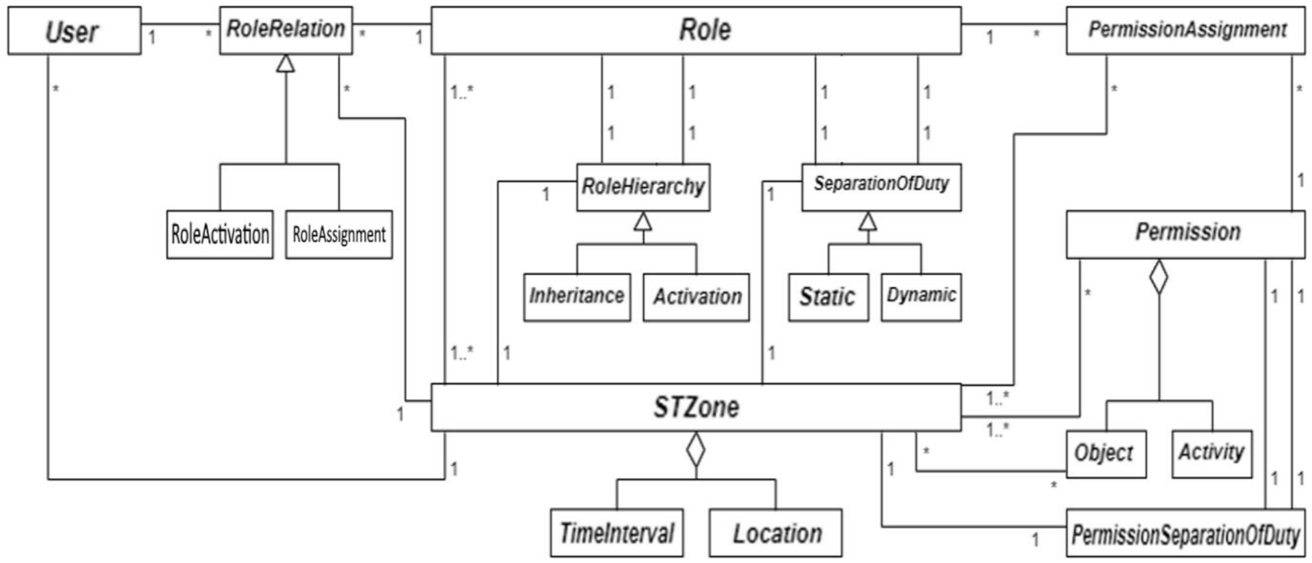
Figure 1: A thief attempts to disable the motion detector using a stolen mobile but is prevented due to location constraints.



GSTRBAC model was designed using the class models of the Unified Modeling Language (UML), the standard modeling language in the software industry [6].

However, the UML class model provides an abstract representation of GSTRBAC policies and lacks descriptions of how these policies are stored, retrieved, and administered.

Figure 2: UML class model representing GSTRBAC policies.



This paper aims to describe the implementation of a database system for storing and administering GSTRBAC policies. To achieve this goal, we first convert the UML policy class model to a relational database schema, which subsequently is used to create policy databases. Second, we use the Structured Query Language (SQL), C# programming language, and the Visual Studio Table Designer tool to define, manipulate, and access a GSTRBAC policy database for a simple company system. Third, we outline a web application developed using the ASP.NET Model View Controller (MVC) framework. The web application is used by access control analysts for the administration of the policy database.

This paper is organized as follows. Section 2 gives an overview of the GSTRBAC model. Conversion of the UML policy class model to a relational database schema is detailed in Section 3. The implementation details of the database in SQL and the web application are discussed in Section 4. Lastly, Section 5 summarizes the paper and points out future work.

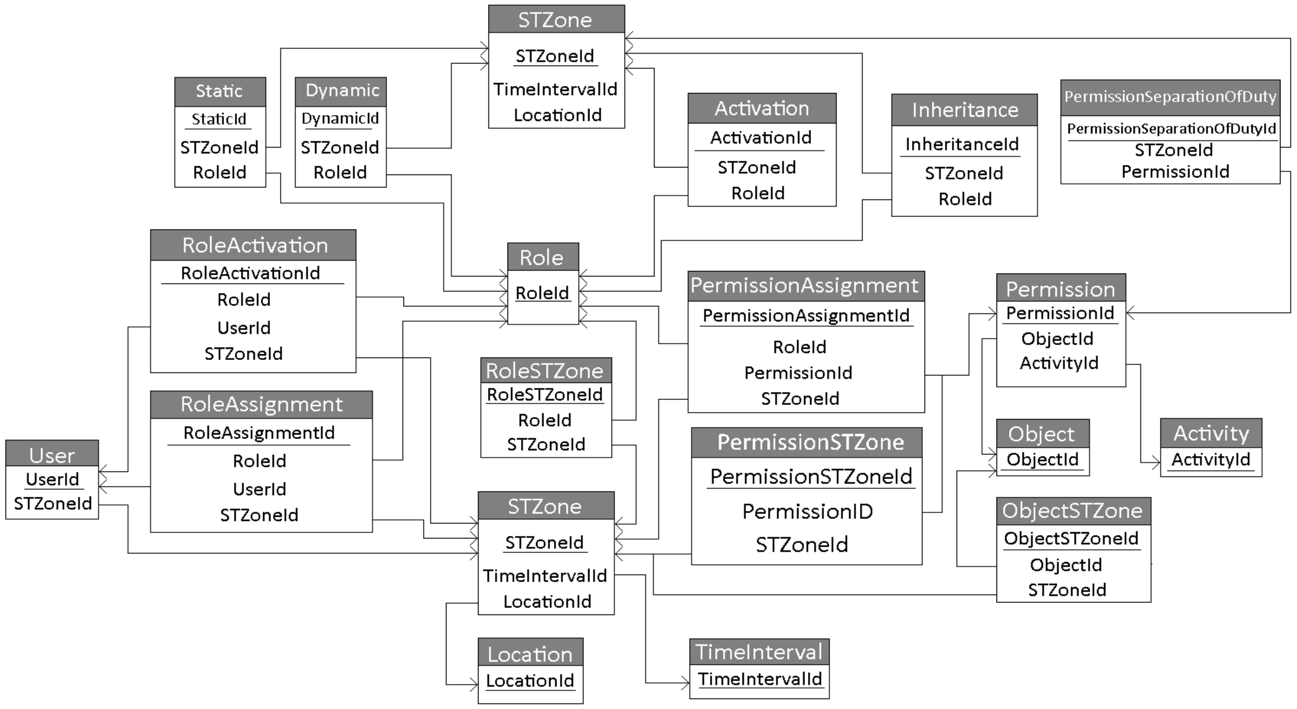
2 Overview of GSTRBAC

Figure 2 depicts the UML class model representing the different components of the GSTRBAC model and their interrelationships with spatio-temporal concepts. The main classes are *User*, *Role*, and *Permission*. In GSTRBAC, class *User* represents a collection of personnel that can access the system's resources. Class *Role* is a collection of jobs that a user can be assigned to, e.g., a teacher. *Permission* is a collection of actions that a *Role* can perform, such as a teacher editing grades. The GSTRBAC main classes are related to each other by the two UML association classes *RoleRelation* and *PermissionAssignment*. Class *RoleRelation* has two subclasses *RoleAssignment* and *RoleActivation*, denoting that a user is assigned to a particular role and a role is activated by a specific user, respectively. Similarly, *PermissionAssignment* indicates the permissions assigned to a role.

The GSTRBAC model also supports other higher-level access control concepts such as role hierarchy (represented by class *RoleHierarchy*) and separation of duty (denoted by class *SeparationOfDuty*). Role Hierarchy indicates a chain of authority for roles and can be of two types *Activation* (a user assigned to a senior role activate a junior role) and *Inheritance* (a user assigned to a senior role can use the permissions for the junior roles). Separation of duty occurs when two users cannot be assigned to two conflicting roles (e.g., the same user should not be assigned to billing clerk and account receivable clerk roles), or a role cannot be assigned to two conflicting permissions (e.g., a loan officer is not permissible to issue loan request and approve it). As shown in the figure, almost all components are associated with the class *STZone* to express the time and the location these components are enabled. Interested readers can find more information in the GSTRBAC model paper [5].

Having gone over the key concepts of GSTRBAC I will briefly explain the methods we used to arrive at the current UML class model. UML follows a simple set of rules that analyze papers to arrive at the final design of a class model. I will use the previous paragraphs in section 2 as an example paper to explain these rules. First according to UML guidelines in an informative or scientific paper we must identify any nouns found in a sentence. For our section 2 example we have nouns such as Role, GSTRBAC, User, Permissions, etc. Once we identify nouns we deem important to a system we then insert them into the class model as a class. These classes are used to represent an object we can manipulate. Following we need to denote the elements of a class we can manage, we call these attributes. In an informative paper such as this one many of the attributes a class may posses have been described. In our case we can determine a permission to have an attribute of an action in its manageable attributes. Many of these attributes will have to be assigned a data type to best fit their representation. Data types being the way we store them within coding format (a name would be a string in coding language). Once we have all the classes attributes we then need to determine the ways in which these classes will interact with each other. In our paper we state a *Role* to have a *Permission*, as such these two classes have a relationship. UML denotes this relationship as a line to connect them. In our case, we added an additional class to act as an intermediary between them. From here it is up to the designers understanding of the system to determine how the various classes will interact.

Figure 3: The relational database schema produced from conversion of GSTRBAC UML class model.



3 GSTRBAC Relational Database

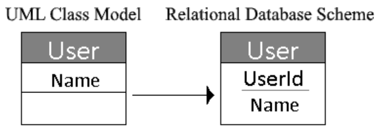
We created a relational database to manage and process the GSTRBAC policies. In this section, we provide the rules we followed to convert the UML design model (i.e. Figure 2) to the relation database schema outlined in Figure 3.

We applied the following set of rules that convert the UML class model to the relational database schema, as suggested by Alvaro Monge [7]:

1. Map UML classes to tables in the relational database schema.
2. Attributes of a UML class are mapped to fields of corresponding tables.
3. Create primary keys for all tables, if not already included in fields.
4. One-to-many associations between classes in UML are represented as foreign keys in the schema diagram.
5. Many-to-many associations between UML classes are represented as a separate table that connect the classes of the associations together.

Applying the rules above, we begin by converting all classes in Figure 2 to schema tables. For example, Figure 4 displays the output of the conversion for the UML class *User*. In the conversion process, we keep the names of UML classes the same for tables (indicated by the gray box in the diagram). After the conversion of the classes, any attributes of the classes will be assigned as fields to the corresponding named tables.

Figure 4: Conversion of an UML class to a relational database table.

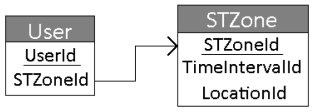


In a database schema, every table must have a primary key; therefore, following the third rule above, we created a primary key for each table. For example, in Figure 4, we added the UserId field to the *User* table (underlined to indicate a primary key).

As shown in the UML class model (Figure 2), the class *STZone* is associated with class *User* by a one-to-many association. Therefore, applying rule 4 above, we indicate the foreign key *STZoneid* in the *User* table.

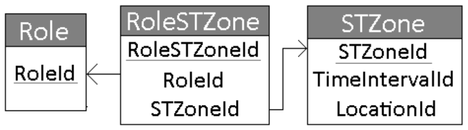
In the schema diagram, a foreign key is denoted by an arrow pointing to the referenced field of another table. Figure 5 shows this mapping.

Figure 5: Mapping example of one-to-many associations.



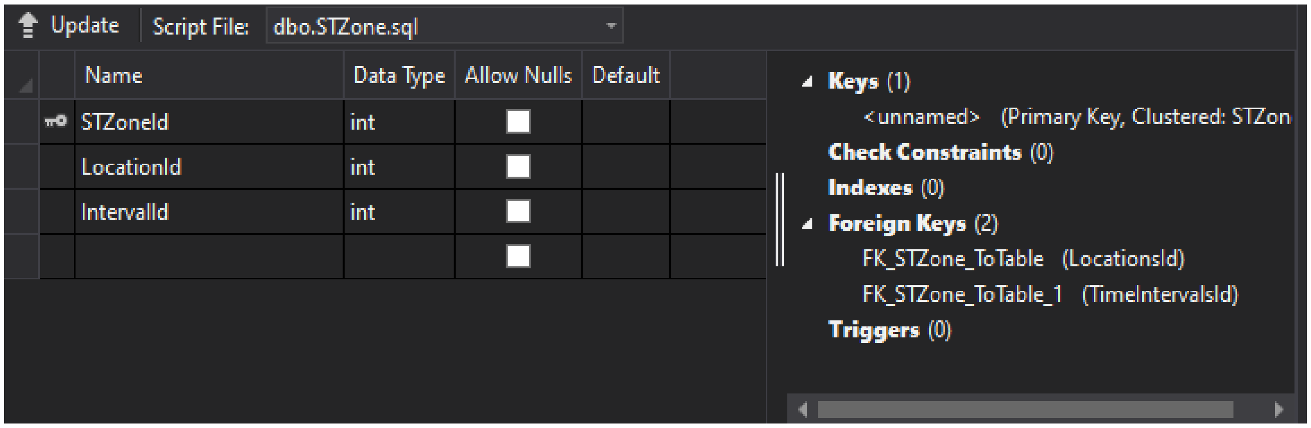
In the case of many-to-many associations, we apply rule 5 to create junction tables to store all the relationship instances between the two classes. For example, the many-to-many association between classes *Role* and *STZone* (in Figure 2) is represented by the *RoleSTZone* junction table between the corresponding *Role* and *STZone* tables, as shown in Figure 6.

Figure 6: A junction table to represent a many-to-many association.



In the GSTRBAC UML class model (Figure 2), *RoleHierarchy* and *SeparationOfDuty* represent generalization-specialization relationships. We convert these UML classes to schema tables by using the foreign keys

Figure 8: Visual Studio Table Designer user interface.



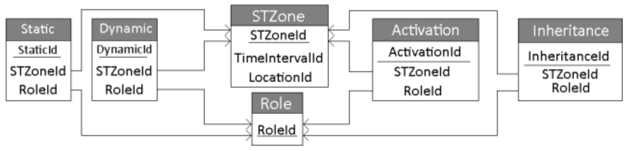
Update Script File: dbo.STZone.sql

	Name	Data Type	Allow Nulls	Default
	STZoneId	int	☐	
	LocationId	int	☐	
	IntervalId	int	☐	
			☐	

Keys (1)
<unnamed> (Primary Key, Clustered: STZon
Check Constraints (0)
Indexes (0)
Foreign Keys (2)
FK_STZone_ToTable (LocationsId)
FK_STZone_ToTable_1 (TimeIntervalsId)
Triggers (0)

of the generalizations in the specialization classes. For example, *SeperationOfDuty* has two specialization classes: *Static* and *Dynamic*. The generalization class has a one-to-many association with the *STZone* and *Role* classes; hence, two foreign keys are used. As a result, in the schema diagram, the specialization classes *Static* and *Dynamic* both contain foreign keys of *STZone* and *Role*, as shown in Figure 7.

Figure 7: How we handle generalization-specialization relationships.



Similarly, we applied the same conversion to the *RoleHierarchy* generalization-specialization relationship.

4 Database Implementation and Administration

To demonstrate the applicability of GSTRBAC policies, we instantiated the policy schema in Figure 3 for a simple company. We followed the three-tier architecture, where the client machine interacts with the database through a separate database server [8]. In this section, we first outline our use of Visual Studio's table designer to create the company policy database. We then overview the ASP.NET Model View Controller (MVC) application that security analysts can use to interact with the policy database for administration purposes.

4.1 Policy Database Development

We used SQL as the database definition and manipulation language in this project. Visual Studio provides a package for SQL Server database development. This package contains the Table Designer tool to create and instantiate databases. We used Table Designer to create tables for the schema in Figure 3. Figure 8 shows the Table Designer's interface that automatically generates the SQL commands to create the *STZone* table from the schema in Figure 3

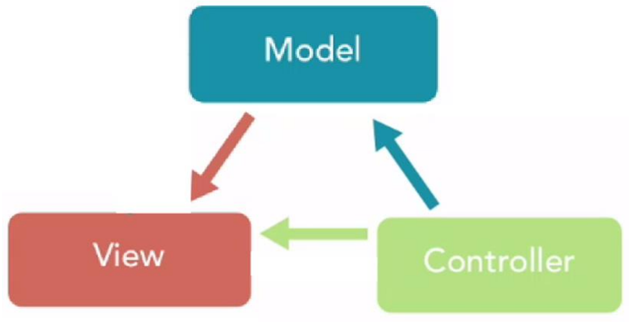
After creating the database tables, we instantiated them with data modeled after a software development company. We represent this data that follows as data sets encapsulating the overall concepts of the GSTRBAC model.

1) Users = {Bob, Ben, Alice, Rachael, Clare, Sam}.
2) Time Intervals = {*i*1, *i*2}. where *i*1 = [8 a.m. to 6 p.m.] and *i*2 = [6 p.m. to 8 a.m.].
3) Locations = {Home, DevelopmentOffice, TestingOffice, DirectorOffice, DepartmentBuilding}.
4) Roles = { Software Engineer (SE), Software Programmer (SP), Test Engineers (TE), Programmer Supervisor (PS), Test Supervisor (TS), Project Lead (PL) }.
5) Objects = { Project Files (*obj*1), Test Files (*obj*2), Programmer Logs (*obj*3), Test Logs (*obj*4), Programmer Supervisor Report (*obj*5), Test Supervisors Reports (*obj*6) }..
6) Activities = {*read*, *write*, *copy*, *run*, *review*}.
7) Permissions = { *P*1(read, *obj*1), *P*2(write, *obj*1), *P*3(copy, *obj*1), *P*4(write, *obj*2), *P*5(run, *obj*2)), *P*6(review, *obj*4), *P*7(review, *obj*3)), *P*8(read, *obj*5) }.
8) STZones = {*z*0, *z*1, *z*2, *z*3, *z*4}, where *z*0 = (DepartmentBuilding, *i*1), *z*1 = (Home, *i*2), *z*2 = (DevelopmentOffice, *i*1), *z*3 = (TestingOffice, *i*1), and *z*4 = (DirectorOffice, *i*1).
9) User Role Assignment = { (Ben, SP, *z*1), (Ben, SP, *z*2), (Bob, PS, *z*2), (Alice, PL, *z*4), (Clare, TS, *z*3), (Rachael, TE, *z*1), (Rachael, TE, *z*3), (Sam, SE, *z*0) }.
10) Permission Role Assignment = { (SP, *P*1, *z*1), (SP, *P*1, *z*2), (SP, *P*2, *z*1), (SP, *P*2, *z*2), (SP, *P*3, *z*2), (TE, *P*4, *z*1), (TE, *P*4, *z*3), (TE, *P*5, *z*3), (TS, *P*6, *z*3), (PS, *P*7, *z*2), (PL, *P*8, *z*4) }.
11) Role Hierarchy = { (PS, SP, *z*2), (TS, TE, *z*3), (PL, PS, *z*0), (PL, TS, *z*0) }.
12) Separation of Duty = { (SP, TE, *z*0) }.

These data sets represent groups of entities that make up the GSTRBAC model; as a result, we have data such as: (1) Users representing the various users that may access company resources. (2) Time Intervals and (3) Locations that have been embedded by the company. (4) Roles that a user may have in the company. These various data sets assist us in establishing information that a potential company may allow administrators to manage. Additionally, they provide data that allows us to test the overall application and its functionality.

Following the instantiation of the data we then begin development of the external web-based application. We follow this process as the web application requires a model of the database. Because of this we needed to clarify the various data types of the information, for instance a user has a name which in code is called a string. For this reason we needed to match the data types of the database information to that of the model required for the web application.

Figure 9: General Representation of the MVC Framework



4.2 Administration Web Application

ASP.NET MVC is a framework for developing web applications in Visual Studio using the C# language [9]. This framework enables the communication between a database and a web application using the Models, Views, and Controllers architecture detailed in figure 9. In the MVC framework, Models represent the structure of a database using classes in C#. We do so as the two other components use this structure to appropriately convey data from a database. First and foremost it accomplishes this by denoting the type of data that will be retrieved or stored in the database. Continuing, the View component displays data from a table found in the database to a web application following the structure of the Model. It does so in conjunction with the Controller and Model components. In which, the Controller passes data to the View where we format it to allow for easier administration. Further, Controllers provide the logic for manipulating data and passing it to the View to display. In the Controller component we develop the complex actions that administrator then use in policy management.

The following paragraphs detail the development of the web application using the MVC framework. Before I begin explaining the details of the development, I need to give context to Language Integrated Queries (LINQ), Razor and HyperText Markup Language (HTML). LINQ is a variant of SQL that C# programs use to query the data from a database and is the primary language used in the Controller component of MVC. Razor and HTML are coding formats used in the development of a web page. HTML is the standard language used in this aspect and is used to determine how a web page will be displayed, the View component uses it frequently. Razor acts similarly to HTML and compresses HTML to allow for improved readability and management. We use Razor in the View component to allow for improved communication between the View and Controller components. Now that I have introduced the different coding languages used in the web application we will be looking at its development. We begin with Listing 1 showing an example of a LINQ query. This query retrieves the information contained in the *User* table.

Listing 1: C# LINQ code that retrieves User table data.

```
var userTable = From  users in _Context.User  Select users;
```

Listing 2 shows a query that updates the *User* table to include a new user instance. This new user is inputted by administrators who can determine the information attributed to a new addition. In figure 4 we allow a user in the relational database schema to have a name. As a result, when administrators include a new user they can input the name of this user and any other information available. Note that listing 2 is the general LINQ code to add a new user, the code needed to allow for such inputs has been left out of this example for simplicity.

Listing 2: A LINQ query that adds a user instance to the *User* table.

```
_context.Add(users);
await _context.SaveChangesAsync();
```

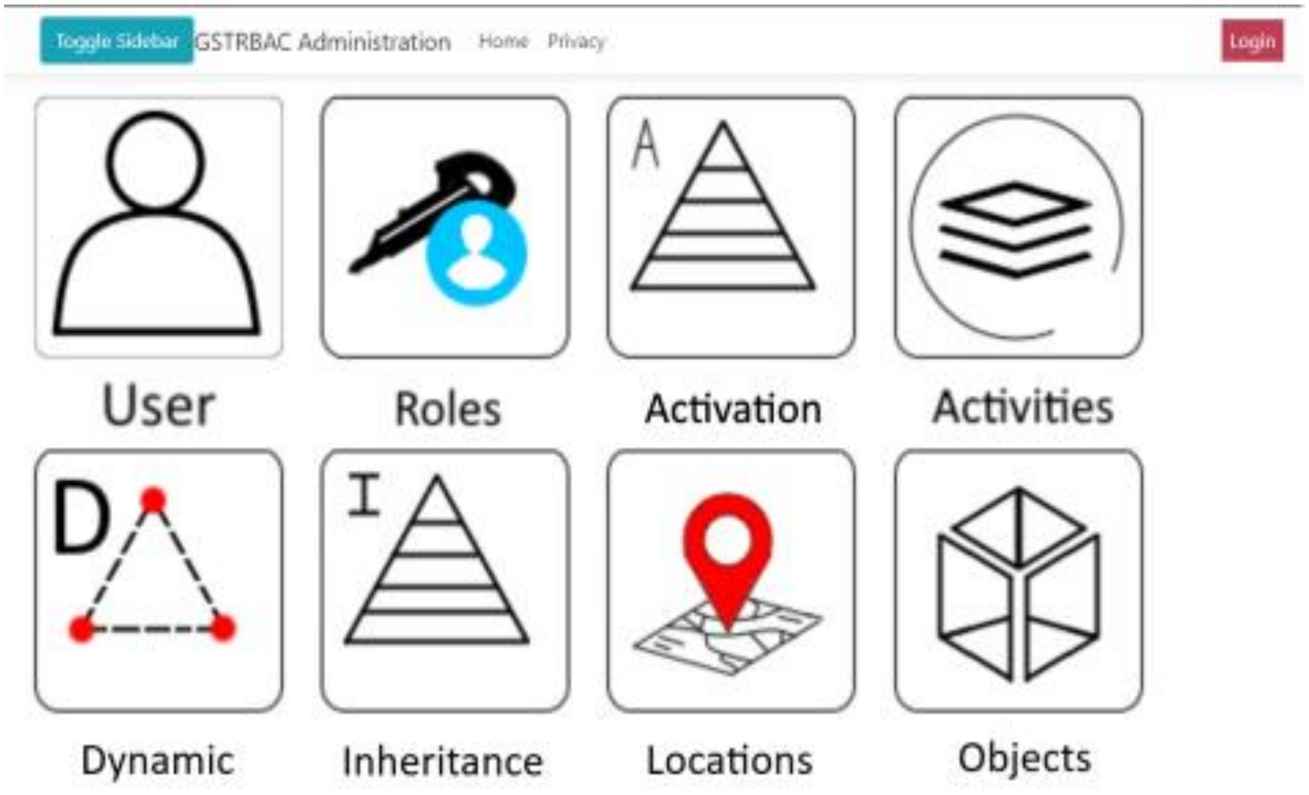
We can combine such queries shown in listings 1 and 2 to enable administrators of the system to perform complex actions. Listing 3 shows an example LINQ code we use to display the roles of a user to an administrator. The query shown for this listing is once more the simplified version to avoid overly complex code or LINQ statements.

Listing 3: Query that retrieves available roles of a user.

```
public async Task<IActionResult> GetAssignedRolesView(int? us-
erSearchId, int? zoneId)
{
    if (userSearchId == null || zoneId == null)
    {
        var roleTable = from roles in _context.Role select roles;

        return View(await roleTable.ToListAsync());
    }

    var assignedRoles = GetAssignedRoles(userSearchId,zoneId);

    return View(await assignedRoles.AsQueryable().ToListAsync());
}
```

The following paragraph will go further in detail regarding listing 3. For this example, we show an action that has been coded into the Controller component of the MVC framework. This action represents a function that has been created under the C# coding and LINQ languages. The first two lines detail the name of the action and the inputs that an administrator can assign when executing the action, these are denoted by the '?' figure in parenthesis. The inputs available to administrators are the userSearch Id and zoneId. Manipulating these input parameters, as described in coding knowledge, administrators can thereby change the information displayed to the View component. We allow such a change using an if statement denoted by line 4's 'if' keyword. In this statement we determine if the administrator has not inputted any information to the parameters and from here can diverge to two separate paths. The first is when either or none of the parameters have been inputted. This results in the full list of roles being viewable to administrators. The second is when both parameters have been given an input. The result is two function calls that return the roles given to the specified user. These LINQ statements allow for the communication between the policy database we previously instantiated with data and the web application itself. It does this by taking any data retrieved from the database and passing it to the View component of the MVC framework. Listing 4 shows an example of the GetAssignedRolesView action in the View component.

Figure 10: The GSTRBAC administration web application directory page.



Listing 4: HTML View page displaying roles of a user.

```
<form asp-controller="Users" asp-action="GetAssignedRolesView">
<p>
  Role: <input type="number" name="userSearchId" />
  Selected STZone: <input type="number" name="zoneId" />
  <input type="submit" value="Filter" />
</p>

</form>
<table class="table">
   <tbody>
      @foreach (var item in Model)
      {
         <tr>
            <th>
               @Html.DisplayFor(modelItem => item.RoleId)
            </th>
            <td>
               @Html.DisplayFor(modelItem => item.Name)
            </td>
            <td>
               @Html.DisplayFor(modelItem => item.Abbreviation)
            </td>
         </tr>
      }
   </tbody>
</table>
```

I will go over the key aspects of the HTML code presented by listing 4 and their importance in displaying information to an administrator. Majority of the statements in the View component are denoted as HTML code that allow for the data to be displayed. We use a more modern approach in the form of Razor statements. These Razor statements can be identified by the '@' symbol. Using multiple Views we allow for administrators to perform a variety of actions. Following I will explain the end result of using the MVC framework.

Figure 10 displays the directory page of the web administration application through which the data in the tables can be manipulated. The web application displays pages through the MVC View component, which enables administrators to pass data to Controllers. The Controllers then use the input in LINQ queries to perform actions predefined by the company's policies.

Thus, administrators can handle tables and perform maintenance on the policy database.

5 Conclusion

The purpose of this paper was to describe the results of an undergraduate research project. The project targeted the implementation of the policy database in a web application for administrating spatio-temporal policies as formalized by the UML GSTRBAC class model. We discussed the rules for the conversion of a UML class model to a relational database schema.

We, then, outlined the tools we used to create a web application for a policy database for use in a simple company administrating GSTRBAC policies.

Future work will focus on the integration of the web application with a software system to enable authorization for users. Additionally, we plan to improve the web application to include protections against common cyber-attacks.

Funding

This work has been supported by the Act On Ideas Program under the Texas A&M International University.

References

[1] Ravi Sandhu, Edward Coyne, Hal Feinstein, and Charles Youman, Role-based access control models. *IEEE Comput*er, 29(2):38–47, 1996.
[2] Indrakshi Ray and Manachai Toahchoodee. A spatio-temporal role-based access control model. *In Proc. 21st Annu. IFIP WG 11.3 Working Conf. Data Appl. Security (DBSec)*, pages 211–226, 2007.
[3] Subhendu Aich, Shamik Sural, and A.K. Majumdar. STARBAC: Spatio temporal role based access control. In *On The Move to Meaningful Internet Systems (OTM)*, pages 1567–1582, Vilamoura, Portugal, 2007.
[4] Arjmand Samuel, Arif Ghafoor, and Elisa Bertino. A framework for specification and verification of generalized spatio-temporal role based access control model. In *Tech. Rep. CERIAS* TR 2007-08, 2007.
[5] Ramadan Abdunabi, Mustafa Al-Lail, Indrakshi Ray, and Rober France. Specification, Validation, and Enforcement of a Generalized Spatio-Temporal Role-Based Access Control Model. *IEEE Systems Journal*, 7(3):501-515, 2013.
[6] Grady Booch, James Rumbaugh, and Ivar Jacobson. *The Unified Modeling Language User Guide*. Addison-Wesley Professional, Reading, MA, 2005.
[7] Alvaro Monge. *Database design with UML and SQL, 4th edition*. https://web.csulb.edu/colleges/coe/cecs/dbdesign/dbdesign.php.
[8] Abraham Silberschatz, Henry Korth, and S. Sudarshan. *Database System Concepts*. McGraw-Hill Education, New York, 2020.
[9] Jon Galloway, Brad Wilson, Scott Allen, and David Matson. *Professional ASP.NET MVC 5*. John Wiley & Sons, Inc., Indianapolis, 2014.