New Approach Targeting Security Patterns Development and Deployment

Azzam Mourad[a], Hadi Otrok[b] and Lama Baajour[a]

[a]Department of Computer Science and Mathematics,
Lebanese American University, Beirut, Lebanon
{azzam.mourad, lama.baajour}@lau.edu.lb

[b]Department of Computer Engineering,
Khalifa University of Science, Technology & Research, Abu Dhabi, UAE
{hadi.otrok}@kustar.ac.ae [*]

Abstract

In this paper, we address the problems related to the applicability and usability of security patterns. In this context, we propose a new approach based on aspect-oriented programming (AOP) for security patterns development, specification and deployment. Our approach allows the security experts to deliver their security patterns that describe the steps and actions required for security solutions, including detailed information on how and where to integrate each one of them. It also provides the pattern users with the capabilities to deploy well-defined security solutions. The pattern users are required to have knowledge in AOP with minimal expertise in the corresponding security solution domain. Moreover, we design and implement the RBAC (Role Based Access Control) model for a Library Circulation system called RBAC-LB. The elaborated RBAC-LB model illustrates all the procedures and mechanisms of the approach phases and provides authentication/access control features for the library system.

1 Motivations & Background

In today's computing world, security takes an increasingly predominant role. The industry is facing challenges in public confidence at the discovery of vulnerabilities, and customers are expecting security to be delivered out of the box. Computer security professionals have been promoting, for many years, tools and best practices guidelines to be used by the software development industry, with little adoption so far. Developers, often pressed by a dominating time-to-market priority, must deal with a large set of technical and non-technical issues, in which case security concerns are not thoroughly addressed. The practical help for developers are typically centered around frameworks, standard and design guidelines, which can be of limited use for both implementers and maintainers.

Security design patterns have been proposed recently as a tool for the improvement of software security during the architecture and design phases [24, 19, 7]. Since the appearance of this research topic in 1997, several catalogs have emerged, and the security pattern community has produced significant contributions, with many related to design. Security design patterns address the problem from a different perspective, by encapsulating expert knowledge in the form of well-defined solutions to common problems. The idea of patterns was introduced by Christopher Alexander et al. [1] in the field of building architecture, and was later reused in the object-oriented world. Security patterns are such patterns, but applied for computer/information security. A security pattern describes a particular recurring security problem that arises in a specific context and presents a well-defined generic scheme for a security solution [21].

However, many problems related to the applicability and usability of security patterns arise. Applying and deploying these patterns still require from the developers to have high level security expertise and to take detailed and critical programming decisions, which contradict with the major intent of security patterns. Developers are typically experts in the functional requirements of the software with a minimal security knowledge, which results into weak security decisions. As a result, elaborating methodologies, which guide the security experts to develop and deliver their security patterns, and at the same time, provide the developers with the capabilities to apply the patterns with minimal security expertise, is becoming a very challenging and interesting issue in this domain of research.

In this paper, we propose a new approach based on

[*]This work is supported by the Lebanese American University and CNRS Lebanon

aspect-oriented programming (AOP) for security patterns development, specification and deployment. The proposed approach consists of two related phases applied independently. From one side, the first phase allows the security experts to deliver their security patterns that describe the steps and actions required for security solutions, including detailed information on how, when and where to integrate each one of them. On the other side, the second phase provides the pattern users with the capabilities to deploy well-defined security solutions. The pattern users are required to have knowledge in AOP, with minimal expertise in the corresponding security solution domain. The main contribution of this methodology is addressing the problems related to the application, usability and integration of security patterns, which are mainly known as applicability problems.

Moreover, our proposition includes the elaboration of a RBAC (Role Based Access Control) model for a Library Circulation system called RBAC-LB. The RBAC-LB model illustrates all the procedures and mechanisms of the approach phases and provides authentication/access control features for the library system. We demonstrate the feasibility of our propositions by elaborating and developing the library system, RBAC-LB model and patterns together with their aspects that realize the proposed approach and RBAC-LB model. Case studies and experimental results are also presented.

The remainder of this paper is organized as follows. In Section 2, we introduce the contributions in the field of security patterns and AOP for software security. Afterwards, we present in Section 3 an overview of the proposed approach. Then, we illustrate in Section 4 the proposed RBAC-LB model for the Library Circulation System. We also describe in this section the elaborated patterns and aspects that realize the proposed approach and RBAC-LB model. Section 5 presents the case studies and experimental results. Finally, we offer concluding remarks in Section 6.

2 Related work

The current research on the topic of security design patterns is characterized by various publications. In [24], Yoder and Barcalow introduced a 7-pattern catalog. In fact, their proposed patterns were not meant to be a comprehensive set of security patterns, rather just as starting point towards a collection of patterns that can help developers address security issues when developing applications. Kienzle et al. [13, 14] have created a 29-pattern security pattern repository, which categorized security patterns as either structural or procedural patterns. Structural patterns are implementable patterns in an application whereas procedural patterns are patterns that were aimed to improve the development process of security-critical software. The presented patterns were implementations of specific web application security policies.

Romanosky [19] introduced another set of design patterns. The discussion however has focused on architectural and procedural guidelines more than security patterns. Brown and Fernandez [7] introduced a single security pattern, the authenticator, which described a general mechanism for providing identification and authentication to a server from a client. Fernandez and Warrier extended this pattern recently in [8]. Braga et al. [4] also investigated security-related patterns specialized for cryptographic operations. They showed how cryptographic transformation over messages could be structured as a composite of instantiations of the cryptographic meta-pattern. The Open Group [2] has proposed a catalog of 13 patterns based on architectural framework standards such as the ISO/IEC 10181 family. The most recent work in this domain is from Schumacher et al. [21]. They offered a list of forty-six patterns applied in different fields of software security, including many previously proposed patterns.

All these approaches and patterns presented interesting and important security solutions. However, their main problems are related to the applicability and usability of security patterns. Applying these patterns still requires the developers to have high level security expertise and to take detailed and critical programming decisions, which contradict with the major intent of security patterns. Developers are typically experts in the functional requirements of the software but know little about security, which results in them not knowing what security mechanisms make sense at which moments and places.

Regarding the use of AOP for security, the following is a brief overview of the available contributions. Cigital labs proposed an AOP language called CSAW [22], which is a small superset of C programming language dedicated to improve the security of C programs. De Win, in his Ph.D. thesis [5], discussed an aspect-oriented approach that allowed the integration of security aspects within applications. It is based on AOSD concepts to specify the behavior code to be merged in the application and the location where this code should be injected. In [3], Ron Bodkin surveyed the security requirements for enterprise applications and described examples of security crosscutting concerns, with a focus on authentication and authorization. Another contribution in AOP security is the Java Security Aspect Library (JSAL), in which Huang et al. [10] introduced and implemented, in AspectJ, a reusable and generic aspect library that provides security functions. In [23], Shlowikowski and Ziekinski discussed some security solutions based on J2EE and JBoss application server, Java Authentication and Authorization service API (JAAS) and Resource Access Decision Facility (RAD). These solutions are implemented in AspectJ. These approaches are useful to explore the feasibility of using AOP in Software security, hence, we can benefit from their achievements to build our security models.

3 Approach Overview

Aspect Oriented Programming (AOP) [18, 9, 6, 12, 11] is one of the most prominent paradigms that have been de-

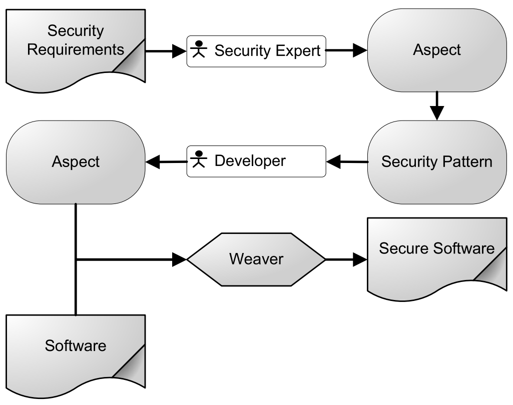


Figure 1. Approach Schema

vised for integrating non-functional requirements (e.g. security) into software. The main objective of AOP is to have a separation of concerns between cross-cutting concerns. This is achieved through the definition of aspects. Each aspect is a separate module in which pointcuts are defined. A pointcut identifies one or more join points. A join point identifies one or many flow points in the program. At these points, some advices will be executed. An advice contains some code that can alter the program behavior at a certain flow point. The integration of aspects within the application code is performed through a weaving technology (e.g., AspectJ [11]).

Aspects allow to precisely and selectively define and integrate security objects, methods and events within application, which make them interesting solutions for many security issues. Many contributions [22, 5, 3, 10], in addition to our experiments [16], have proven the usefulness of AOP for integrating security features into software. In this context, we present in this section a methodology to develop the security solutions and their corresponding aspects and patterns based on AOP. This methodology will also allow the developers to integrate and deploy the patterns by implementing their corresponding aspects and weaving them into their software. Such approach contributes in solving the problems related to the application and integration of security patterns, which is mainly known as applicability problems. Although our solution cannot be applied on all type of patterns, it can solve big part of the aforementioned problems. Figure 1 illustrates the proposed methodology.

The methodology is composed of two related phases applied independently. The first one should be performed by security experts and consists of developing the security solutions and their corresponding aspects and patterns. First, the security experts develop the security solutions, integrate them manually (i.e., write their code in the original program) into the software and verify their correctness with respect to the security requirements provided. Afterwards, they develop the aspects that dictate systematically all the steps performed when manually securing the software (See Listings 1, 2, 3, 4 and 5). Then, the aspects should be merged with the original software by using one of the available weavers (e.g. AspectJ for Java, AspectC++ for C/C++). Finally, once the resulted software is verified for security correctness, the corresponding security patterns can be elaborated and specified in aspect-oriented manner (See Listings 6 and 7).

The second phase is performed consequently by the users of the provided security patterns. It consists of deriving and implementing the aspects corresponding to the selected security patterns and then weaving them into the software. The developers are required to have knowledge in AOP with minimal security expertise. Moreover, since the security solutions provided in the patterns were already developed using AOP, then reversing and applying the procedures are feasible(i.e., deriving the aspects from the patterns, implementing them and then integrating them into applications).

4 RBAC-LB: AN AUTHORIZATION MODEL FOR A LIBRARY SYSTEM

In this section, we focus on describing our library system and elaborating its corresponding role based authorization model by following our proposed approach. We first present an overview of the library system and describe the proposed RBAC-LB model. Afterwards, we present aspects and patterns realizing the proposed approach and model.

4.1 Library System Overview

Our system (illustrated in Figure 2) is a Library Management System, in particular, the Circulation division in the Library. We developed four of the main services of Circulation division in addition to a graphical user interface to interact with the user, get the input and show the results. A user can access one of the services in the system's main page. First, the View Info service allows him/her to view his/her personal information (Name, ID, Date of Birth, Telephone Number, User Type, Address, and Email). Second, Add/Delete library User service allows him/her to add or delete a user to/from the library. Third, Add/Delete Library Item is a service which allows a user to add or delete a new library item to/from the library. Finally, a user may access the Search/Borrow A Library Item service, which allows him/her to search for a library item and borrow it if its available in the library. Each one of the users who can access the system has a record in the database. In other words, each user has an ID and a Password stored in the database in addition to his/her personal information. Moreover, the system database contains a record for each library item which holds information for the following fields: ISBN, type, subject, title, author, and status. We extended our library system by elaborating authentication and access control features. The extension is explored in Figure 2.

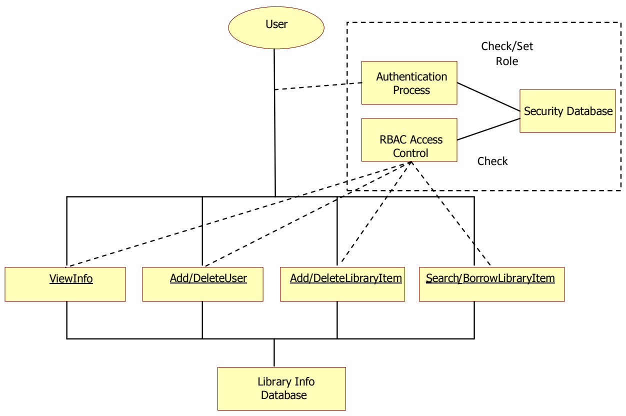


Figure 2. Library System Architecture

4.2 RBAC-LB Model Definitions

This model inherits all the components of the traditional RBAC model such as: users, roles, permissions, role hierarchies, user-role assignment, and role-permission assignment relations. Users are assigned to roles and roles are assigned to permissions. An RBAC permission represents the ability to access a certain system's service. A user is permitted to execute a service activity if he/she is assigned to a role that has the permission to perform that activity. RBAC roles are structured in a hierarchy. Role hierarchies define inheritance relations among the roles in terms of permissions and user assignments. If role r_1 inherits role r_2, then all permissions of r_2 are also permissions of r_1 and all users of r_1 are also users of r_2. More detail about describing RBAC model for web services is presented in [17]. The following is the list of notations:
A: is the identifier of an activity (e.g. Add a new book to the library shelves).
Action: is the type of action that can be performed on activity A.
R: is the set of roles (e.g.: Admin, Staff, Student).
U: is the set of potential users.
P: is the set of permissions, i.e. execution of an activity (e.g. execution of add/delete library user).

To render our RBAC model specification open to future extensions, we will start with a general definition of the main components of the RBAC model.

Definition 1: RBAC-LB Permission

Let LB be our system. An RBAC-LB permission is a tuple (A_i, Action) where A_i is the identifier of an activity in LB and *Action* identifies the type of action that can be performed on activity A_i. For example, the tuple (Add library user, execute) allows the authorized user to run the "Add library service" that is provided by the Library Management System.

Definition 2: RBAC-LB Role

An RBAC-LB role r is a set of attribute conditions $r = \{ac_i \mid ac_i = AttrName_i \ op \ AttrValue_i\}$, where $AttrName_i$ identifies a user attribute name, *op* is a comparison or a set operator, and $AttrValue_i$ is a value, a set, or a range of attribute values.

Two roles r and r' might be identified by the same set of attribute names. However, it is a must that at least one of the values of the attributes of r and r' must be different. A user can be assigned to only one role while two users identified by the same attributes with the same values are assigned to the same role since we assume that a set of attribute conditions uniquely identifies a role.

Definition 3: RBAC-LB Role Hierarchy

Let R be a partially ordered set of roles. A role hierarchy defined over R is the graph of the partial-order relation between the roles in R. If r, $r_0 \in R$ and $r < r_0$, then we say r_0 dominates r. For instance, our library system consists of eight different roles. The most senior role is the $Admin$ which dominates the roles $Staff$ and $Student$. $Staff$, in turn, dominates $Faculty$ and $AdminstrativeStaff$, and $Student$ role dominates $UndergradStudent$, $GradStudent$, and $Alumni$. For example, a user that wants to be assigned to the $Admin$ role must provide digital credentials containing an $Employment$ attribute to $Admin$. Figure 3 and Table 1 illustrate the role hierarchy for the library management system.

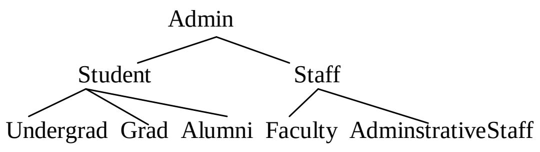


Figure 3. RBAC-LB Role Hierarchy

Table 1. RBAC-LB Role Hierarchy

R	
Admin	{ Employment= Admin, ID= integer of 9 digits, Password= a string of at most 9 characters}
Staff	{ Employment= Staff, ID= integer of 9 digits, Password= a string of at most 9 characters}
Student	{Employment= Student, ID= integer of 9 digits, Password= a string of at most 9 characters}

Definition 4: RBAC-LB User-Role Assignment Relation

Let U be the set of all potential users and R be a partially ordered set of roles. The RBAC-LB user assignment relation is the set of tuples UA= {(u,r) ∈ U x R | ∀ ac_i = $AttrName_i$ op $AttrValue_i$ ∈ r, ∃ $attr_j$ ∈ CredSet(u)[1] | $attr_j$= $AttrName_i$ $\bigwedge$ ac_i is evaluated to true according to value of $attr_j$}. As for the library system, all university members can be considered as potential users and the set of roles is R={ *Admin*, *Staff*, *AdminstrativeStaff*, *Faculty*, *Student*, *Alumni*, *Grad*, *UnderGrad*}. Assigning users to roles results in a set of tuples which define the RBAC-Library user assignment relation. For example, in our library system the set of attribute conditions for the Admin role is r= {Type= "Admin", ID= an integer of 9 digits, Password= a string of at most 9 characters}; thus a credential set of the user u={Type= "Admin", ID= "100100100", Password= "100100"} will be evaluated as true and u is assigned to *Admin*.

Definition 5: RBAC-LB User-Permission Assignment

Let P be the set of permissions of the form (A_1, A_2) where A_1 is the activity supported by the system, and A_2 is the action to be executed. Moreover, let RP be the set of permission role assignment. Thus, the RBAC- LB user- permission assignment relation is the set of tuples UP= {(u,p) ∈ U x P | ∃ (u,r) ∈ UA | (r,p) ∈ RP}. For instance, a permission of adding or deleting a library user is, of the form ("Add/Delete Library user", execute), assigned to *Admin* by the RP relation. Thus a user u can add/delete a library user only if he is assigned first to *Admin*.

4.3 Aspects Realizing the RBAC-LB Model

In this section, we describe the authentication and access control aspects realizing the aforementioned RBAC-LB model. The aspects are developed using AspectJ.

Authentication Aspect

Listing 1 illustrates the aspect to authenticate the user and assign him role and permission(s). Role and permission descriptions are presented in the aforementioned RBAC-LB model (please see description in Section 4). We define the pointcut P_1 as the main class that initiates the application. We insert the authentication verification before the execution of the main class, i.e. at the beginning of the main method of the initial class. The authentication verification code consists of a connection string, which specifies information about the data source used; in our case it is the SQL Server 2005. Then, a connection is established with the server to be able to send SQL commands and receive answers. Once the connection is opened with the server, the SQL query "SELECT", which counts the number of rows that matches with the entered *ID* and *password*, is executed. If the number of matching rows is a 1, the user is authenticated and granted access to the system's main services. However, if the returned result is a 0, then the user receives a pop up window indicating authentication failure.

Listing 1. Aspect for Authentication

```
Aspect Authentication
{
Pointcut P1: MainClass;

Before execution P1
{
    string connectionString = "Data Source=LAMLOUM-\\
         SQLEXPRESS;Initial Catalog=
         LibraryCirculation;Integrated Security= TRUE
         ;";
    SqlConnection conn = new SqlConnection(
         connectionString);
    string select = "SELECT COUNT(*) FROM
         LoginCirculation WHERE ID=@id AND Password=
         @password";
    SqlCommand cmd = new SqlCommand(select, conn);
    cmd.Parameters.Add("@id", IDTextBox.Text);
    cmd.Parameters.Add("@password", PasswordTextBox.
         Text);
    conn.Open();
    //when we're returning number of rows
    int successRows = (int)cmd.ExecuteScalar();
    if (successRows == 1)
    {
        MessageLabel.Text = "Authentication SUCCEEDED
            !";
        string url = "Main.aspx?ID=" + IDTextBox.Text
            ;
        Response.Redirect(url);
    }
    else
    {
        MessageLabel.Text = "Authentication FAILED!
             Invalid Username and/or password.";
    }
    conn.Close();
}
```

Access Control Aspect for Add-Delete-User

Listing 2 illustrates the aspect to authorize the user access to the *AddDeleteUser* service. As described in the aforementioned RBAC-LB model (please see description in Section 4), each user is assigned a role together with its corresponding permission(s). We define the pointcut P_2 as the method responsible of adding and deleting a user. We insert the access control verification before the execution of the *AddDeleteUser* method (i.e. at the beginning of the method declaration). The access control verification code consists of a connection string, which specifies information about the data source used; in our case it is the SQL Server 2005. Then, a connection is established with the server to be able to send SQL commands and receive answers. Once the connection is opened with the server, the SQL query "SELECT", which read the type of logged on user to check if he is allowed access to "Add-Delete-User" service, is executed. If the type read is an "Admin", the user is directed to the service access page. However, if the returned types are "Staff" or "Student", then the user receives a pop up window indicating access denial.

Access Control Aspect for Add-Delete-Item

Listing 3 illustrates the aspect to authorize the user access to the *AddDeleteItem* service. We define the pointcut P_3 as

Listing 2. Aspect for Add-Delete-User Access Control

```
Aspect Add-Delete-User-AccessControl
{
Pointcut P2: AddDeleteUser;

Before execution P2
{
    string connectionString = "Data Source=LAMLOUM-\\
         SQLEXPRESS;Initial Catalog=
         LibraryCirculation;Integrated Security= TRUE
         ;";
    SqlConnection conn = new SqlConnection(
         connectionString);
    string select = "SELECT Type FROM
         LoginCirculation WHERE ID=@id";
    SqlCommand cmd = new SqlCommand(select, conn);
    ID = Request.QueryString["ID"].ToString();
    cmd.Parameters.Add("@id", ID);
    SqlDataReader reader;
    conn.Open();
    reader = cmd.ExecuteReader();
    while (reader.Read())
    {
        type2 = reader["Type"].ToString();
    }
    string url3 = "AddDeleteUser.aspx?ID=" + ID;
    if (type2 == "Admin")
        Response.Redirect(url3);
    if (type2 == "Staff")
        ScriptManager.RegisterStartupScript(this,
             typeof(Page), Guid.NewGuid().ToString(),
              "alert('ACCESS DENIED: You are NOT
             ALLOWED TO ACCESS THIS PAGE');", true);
    if (type2 == "Student")
        ScriptManager.RegisterStartupScript(this,
             typeof(Page), Guid.NewGuid().ToString(),
              "alert('ACCESS DENIED: You are NOT
             ALLOWED TO ACCESS THIS PAGE');", true);
    reader.Close();
    conn.Close();
}
```

Listing 3. Aspect for Add-Delete-Item Access Control

```
Aspect Add-Delete-Item
{
Pointcut P3: AddDeleteItem;

Before execution P3
{
    string connectionString = "Data Source=LAMLOUM-\\
         SQLEXPRESS;Initial Catalog=
         LibraryCirculation;Integrated Security= TRUE
         ;";
    SqlConnection conn = new SqlConnection(
         connectionString);
    string select = "SELECT Type FROM
         LoginCirculation WHERE ID=@id";
    SqlCommand cmd = new SqlCommand(select, conn);
    ID = Request.QueryString["ID"].ToString();
    cmd.Parameters.Add("@id", ID);
    SqlDataReader reader;
    conn.Open();
    reader = cmd.ExecuteReader();
    while (reader.Read())
    {
        type5 = reader["Type"].ToString();
    }
    string url4 = "AddDeleteLibraryItem.aspx?ID=" +
         ID;
    if (type5 == "Admin")
        Response.Redirect(url4);
    if (type5 == "Staff")
        Response.Redirect(url4);
    if (type5 == "Student")
        ScriptManager.RegisterStartupScript(this,
             typeof(Page), Guid.NewGuid().ToString(),
              "alert('ACCESS DENIED: You are NOT
             ALLOWED TO ACCESS THIS PAGE');", true);
    reader.Close();
    conn.Close();
}
```

the method responsible of adding and deleting an item. We insert the access control verification before the execution of the *AddDeleteItem* method (i.e. at the beginning of the method declaration). The access control verification code consists of a connection string, which specifies information about the data source used; in our case it is the SQL Server 2005. Then, a connection is established with the server to be able to send SQL commands and receive answers. Once the connection is opened with the server, the SQL query “SELECT”, which read the type of logged on user to check if he is allowed access to ”Add-Delete-Item” service, is executed. If the type read is an “Admin” or “Staff”, the user is directed to the service's access page. However, if the returned type is “Student”, then the user receives a pop up window indicating access denial.

Access Control Aspect for Search-Borrow-Item

Listing 4 illustrates the aspect to authorize the user access to the *SearchBorrowItem* service. We define the pointcut P_4 as the method responsible of searching and borrowing an item. We insert the access control verification before the execution of the *SearchBorrowItem* method (i.e. at the beginning of the method declaration). The access control verification code consists of a connection string which specifies information about the data source used. In our case it is the SQL Server 2005. Then, a connection is established with the server to be able to send SQL commands and receive answers. Once the connection is opened with the server, the SQL query “SELECT”, which read the type of logged on user to check if he is allowed access to “Search-Borrow-Item” service, is executed. This service can be accessed by all users, thus whatever is the type read the user is directed to the service access page.

Access Control Aspect for View-Info

Listing 5 illustrates the aspect to authorize the user access to the $ViewInfo$ service. We define the pointcut P_5 as the method responsible of viewing information. We insert the access control verification before the execution of the $ViewInfo$ method (i.e. at the beginning of the method declaration). The access control verification code consists of a connection string which specifies information about the data source used; in our case it is the SQL Server 2005.

Listing 4. Aspect for Search-Borrow-Item Access Control

```
Aspect Search-Borrow-Item
{
Pointcut P4: SearchBorrowItem;

Before execution P4
{
    string connectionString = "Data Source=LAMLOUM-\\
         SQLEXPRESS;Initial Catalog=
         LibraryCirculation;Integrated Security= TRUE
         ;";
    SqlConnection conn = new SqlConnection(
         connectionString);
    string select = "SELECT Type FROM
         LoginCirculation WHERE ID=@id";
    SqlCommand cmd = new SqlCommand(select, conn);
    ID = Request.QueryString["ID"].ToString();
    cmd.Parameters.Add("@id", ID);
    SqlDataReader reader;
    conn.Open();
    reader = cmd.ExecuteReader();
    while (reader.Read())
    {
        type4 = reader["Type"].ToString();
    }
    string url5 = "SearchBorrowItem.aspx?ID=" + ID;
    if (type4 == "Admin")
        Response.Redirect(url5);
    if (type4 == "Staff")
        Response.Redirect(url5);
    if (type4 == "Student")
        Response.Redirect(url5);
    reader.Close();
    conn.Close();
}
```

Listing 5. Aspect for View-Info Access Control

```
Aspect View-Info
{
Pointcut P5: ViewInfo;

Before execution P5
{
    string connectionString = "Data Source=LAMLOUM-\\
         SQLEXPRESS;Initial Catalog=
         LibraryCirculation;Integrated Security= TRUE
         ;";
    SqlConnection conn = new SqlConnection(
         connectionString);
    string select = "SELECT Type FROM
         LoginCirculation WHERE ID=@id";
    SqlCommand cmd = new SqlCommand(select, conn);
    ID = Request.QueryString["ID"].ToString();
    cmd.Parameters.Add("@id",ID);
    SqlDataReader reader;
    conn.Open();
    reader = cmd.ExecuteReader();
    while (reader.Read())
    {
        type1 = reader["Type"].ToString();
    }
    string url2 = "ViewPersonalInfo.aspx?ID=" + ID;
    if (type1 == "Admin")
        Response.Redirect(url2);
    if (type1 == "Staff")
        Response.Redirect(url2);
    if (type1 == "Student")
        Response.Redirect(url2);
    reader.Close();
    conn.Close();
}
```

Then, a connection is established with the server to be able to send SQL commands and receive answers. Once the connection is opened with the server, the SQL query "SELECT" which, read the type of logged on user to check if he is allowed access to "View-Info" service, is executed. This service can be accessed by all users, thus whatever is the type read the user is directed to the service access page.

4.4 Patterns Realizing the RBAC-LB Model

In this section, we describe the authentication and access control patterns realizing the aforementioned RBAC-LB model. The patterns are specified using a high-level and free form syntax that explains the solution steps to be performed. Different forms of pattern representation can be adopted (e.g. using SHL presented in ([15]). Patterns are structured documentations that capture well-defined solutions to recurring problems. The basic idea is to write down best practices and lessons learned from a given problem domain in an organized way [20, 21]. Christopher Alexander *et al.* [1] provided the first definition of a pattern and its structure in the field of building architecture, which was later reused in the object-oriented world:

"Each pattern describes a problem, which occurs over and over again in our environment, and then describes the core of the solution to that problem...".

"Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution".

The main concepts of the pattern idea are explored in the aforementioned definitions. The core elements of such template are the *name*, *context*, *problem* and *solution*. Other elements could be added to give more detail and explanation on the pattern if needed. The following is a brief description of these elements:

- *Name*: The name of the pattern is the common-usage short expression that encapsulates the pattern meaning. The name is determined and assigned according to the community to which the pattern belongs. It should be expressive, reflect clearly the content of the pattern, and easy to refer to.
- *Context*: The context describes the environment where the pattern could be applied. In other words, it explores when and where the pattern will work and the preconditions under which the problem and its solution appear.
- *Problem*: The problem is a short description of the problem that this pattern aims at solving. It describes the goals and objectives a pattern needs to achieve.

- *Solution*: The solution is a textual description of the pattern that solves the problem. It is presented as a proven solution of the problem proposed by domain experts. Proven solution means that this solution worked at least once in a well defined environment. The level of abstraction of the solution is directly related to the type of the problem. For instance, we can find some patterns that provide design level solutions, while others provide code level solutions. Typically, a problem can have more than one solution, and the best one is only determined by the context where the problem occurs.

Other forms for pattern solution description could be used. For instance, a programming-independent language called *SHL* has been proposed in [15] for describing patterns based on Aspect-Oriented programming. SHL could be used in our methodology to describe the solution part of our patterns instead of the English description.

User Authentication Pattern

Listing 6 illustrates the pattern to authenticate the user and assign him role and permission(s). The authentication verification should be integrated before the execution of the main class of the system. Once the user's ID and Password are entered, a connection to the database will be established to verify the entered values with the available records. If the user is authenticated, role(s) and permission(s) will be assigned to him and the program will be redirected to the main code of the application. On the other hand, if the user is not authenticated, he will be denied access. Here the program will either exit or restart from the initial point.

Listing 6. Pattern for User Authentication

```
Pattern User-Authentication
{
Name
User-Authentication

Context
The application was designed for direct and/or remote
     usage, but the users are not authenticated
     before accessing the application.

Problem
Unauthorized users can access the system, read
     information and temper with the data.

Solution
Before entering the system
    - Get ID and Password
    - Connect to Security Database
    - Check if User is authenticated
    - If authenticated proceeds with main code
    - If not authenticated denies access
}
```

Service Access Control Pattern

Listing 7 illustrates the pattern to authorize the user access to services. The access control verification should be integrated before the execution of the called service. Once the service is called, a connection to the database will be established to verify if the role of the user has permission to access the selected service. If the user is authorized, then he will be redirected to the code of the selected service. On the other hand, if the user is not authorized, he will be denied access. Here the program will either exit or go back to the point before calling the service.

Listing 7. Pattern for Service Access Control

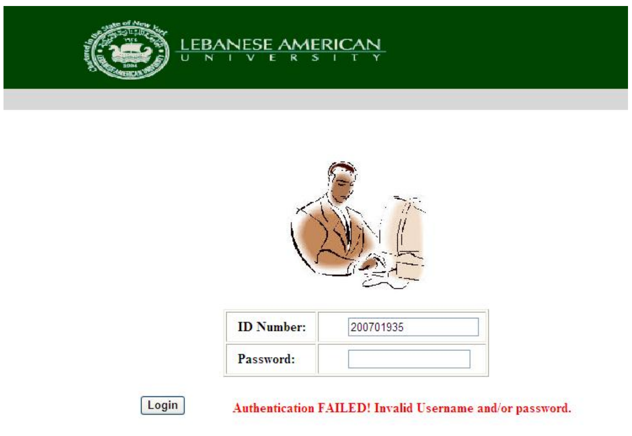
```
Pattern Service-AccessControl
{
Name
Service-AccessControl

Context
The application was designed for direct and/or remote
      usage. The users are authenticated at this
     stage and a role based on RBAC-LB is associated
     to each user. But the users are not yet verified
      if they have the right permission before
     accessing the selected service.

Problem
Users not permitted can access a service provided by
     the application

Solution
Before Accessing the Service
{
    - Connect to Security Database and read the role
        of the user
    - Check if the role of the user (described in the
         RBAC-LB model: only admin has access) has
        permission to access the selected service
    - If authorized proceeds to the service page/code
    - If not authorized denies access
}
```

5 Discussion and Experimental Results

Based on the proposed approach presented in Section 3, we performed each of the following ordered actions:

- We developed and integrated manually (i.e., write their code in the original program) all the RBAC-LB security features in the Library system code.
- We developed and weaved the RBAC-LB aspects of Section 4 in the Library system code.
- We verified practically using testing that integrating the RBAC-LB security features using AOP provide the same solution as integrating them manually. We also verified that such method of deployment did not alter the original functionalities of the library system by testing them before and after integrating security.
- We elaborated the patterns corresponding to the RBAC-LB aspects.

Verifying the successful deployment of the RBAC-LB security features in the original code of the Library system application has been done through extensive testing.



Authentication FAILED! Invalid Username and/or password.

Figure 4. Authentication Failed

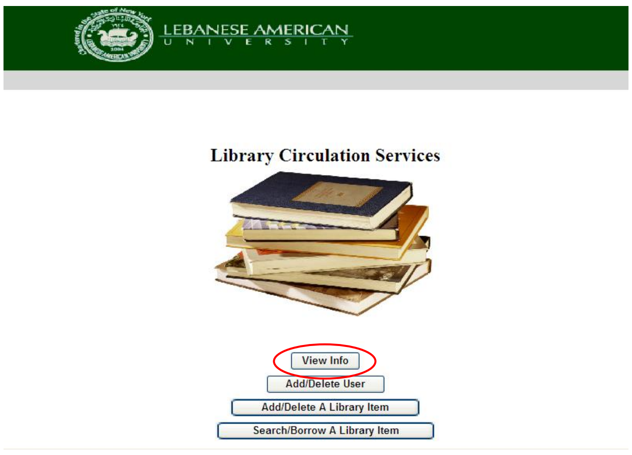
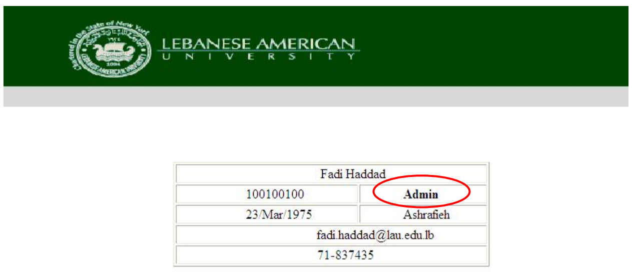

LEBANESE AMERICAN UNIVERSITY
Library Circulation Services
View Info
Add/Delete User
Add/Delete A Library Item
Search/Borrow A Library Item

Figure 5. Authentication Succeed

Other security verification methods could be applied (e.g., formal verification, static/dynamic analysis, requirements test cases, etc.). Additional efforts have been spent on verifying that the original functionalities of the applications have not been altered.

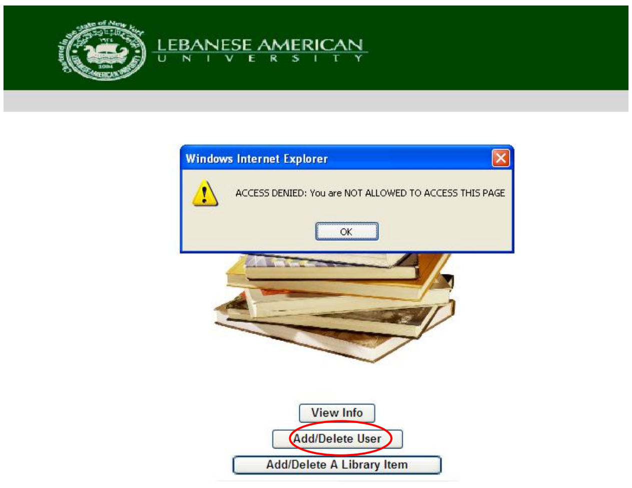
On the other side, the developer using the provided pattern should apply the last three steps in reverse order. He should derive and develop the AOP aspects from their corresponding patterns. Then, he should weave the aspects with the original code of the application. This procedure is feasible because the solutions provided in the pattern were originally derived by the security experts from one or several aspects.

Figures 4, 5, 6 and 7 illustrate screen shots for different execution scenarios of the library system. Figure 4 shows a scenario where the user is not authenticated. The username and/or the password are wrong. An Authentication Failed message has been displayed. Figure 5 shows a scenario where the user is successfully authenticated. The list of services has been displayed. Figure 6 shows a scenario where an authenticated user is accessing successfully a service from the provided list. The selected service has been displayed. Figure 7 shows a scenario where an authenticated user is trying to access a service not allowed to use. An Access Denied message has been displayed.



Fadi Haddad	
100100100	Admin
23/Mar/1975	Ashrafieh
fadi.haddad@lau.edu.lb	
71-837435	

Figure 6. Access Allowed


LEBANESE AMERICAN UNIVERSITY
Windows Internet Explorer
OK
View Info
Add/Delete User
Add/Delete A Library Item

ACCESS DENIED: You are NOT ALLOWED TO ACCESS THIS PAGE

Figure 7. Access Denied

6 Conclusion

We presented in this paper a methodology based on AOP for security patterns development, specification and deployment. The proposed approach consists of two phases. The first phase allowed the security experts to deliver their security patterns, while the second phase provided the pattern users with the capabilities to deploy well-defined security solutions. The requirements are AOP knowledge and minimal expertise in the domain of the applied security solutions. Our proposition addressed the problems related to the applicability and usability of security patterns. Moreover, we proposed the RBAC (Role Based Access Control) model for a Library Circulation system called RBAC-LB. All the procedures and mechanisms of the approach phases are depicted in the RBAC-LB to provide authentication and access control features. The elaborated library system, RBAC-LB model, patterns, aspects and experimental results realize the proposed approach and RBAC-LB model and illustrate their relevance and feasibility.

References

[1] C. Alexander, S. Ishikawa, and M. Silverstein. *A Pattern Language*. Oxford University, 1977.

[2] B. Blakley, C. Heath, and members of The Open Group Security Forum. Security design patterns. Technical Report G031, Open Group, 2004.

[3] R. Bodkin. Enterprise security aspects. In *Proceedings of the Workshop on AOSD Technology for Application-level Security (AOSD04:AOSDSEC)*, 2004.

[4] A. M. Braga, C. M. F. Rubira, and R. Dahab. Tropyc: A pattern language for cryprographic software. Technical Report IC-99-03, Institute of Computing, UNICAMP, Jan. 1999.

[5] B. DeWin. *Engineering Application Level Security through Aspect Oriented Software Development*. PhD thesis, Katholieke Universiteit Leuven, 2004.

[6] J. Evermann. A Meta-Level Specification and Profile for AspectJ in UML. *Journal of Object Technology*, 6(7):27–49, 2007.

[7] F. Lee Brown Jr., J. DiVietri, G. D. de Villegas, and E. B. Fernandez. The authenticator pattern. In *Proceedings of the 6th Annual Conference on the Pattern Languages of Programs (PLoP99)*, 1999.

[8] E. B. Fernandez and R. Warrier. Remote authenticator/authorizer. In *Proceedings of the PLoP 2003*.

[9] L. Fuentes and P. Sanchez. Elaborating UML 2.0 Profiles for AO Design. In *Proceedings of the International Workshop on Aspect-Oriented Modeling*, 2006.

[10] M. Huang, C. Wang, and L. Zhang. Toward a reusable and generic security aspect library. In *Proceedings of the Workshop on AOSD Technology for Application-level Security (AOSD04:AOSDSEC)*, 2004.

[11] G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. Griswold. Overview of AspectJ. In *Proceedings of the 15th European Conference ECOOP 2001*. Springer Verlag, 2001.

[12] G. Kiczales, J. Lamping, A. Menhdhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In M. Akşit and S. Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.

[13] D. M. Kienzle and M. C. Elder. Final technical report: Security patterns for web application development. Technical Report DARPA Contract # F30602-01-C-0164, 2002. `http://www.modsecurity.org/archive/securitypatterns/dmdj_final_report.pdf` (accessed 2008/08/08).

[14] D. M. Kienzle, M. C. Elder, D. Tyree, and J. Edwards-Hewitt. Security patterns repository, 2002. Available at `http://www.modsecurity.org/archive/securitypatterns/dmdj_repository.pdf` (accessed on 2008/11/11).

[15] A. Mourad, M.-A. Laverdière, and M. Debbabi. A high-level aspect-oriented based framework for software security hardening. *Information Security Journal: A Global Perspective*, 17(2):56–74, 2008.

[16] A. Mourad, M.-A. Laverdière, and M. Debbabi. Towards an aspect oriented approach for the security hardening of code. *Computers & Security*, 27(3-4):101–114, 2008.

[17] F. Paci, E. Bertino, and J. Crampton. An Access-Control Framework for WS-BPEL. *International Journal of Web Services Research*, 5(3):20–43, 2008.

[18] J. Pavlich-Mariscal, L. Michel, and S. Demurjian. Enhancing UML to Model Custom Security Aspects. In *Proceedings of the 11th International Workshop on Aspect-Oriented Modeling (AOM@AOSD'07)*, 2007.

[19] S. Romanosky. Security design patterns part 1, 2001. Available at `http://www.romanosky.net/` (accessed on 2008/11/11).

[20] M. Schumacher. *Security Engineering with Patterns*. Springer, 2003.

[21] M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns: Integrating Security and Systems Engineering*. Wiley, 2006.

[22] V. Shah. An aspect-oriented security assurance solution. Technical Report AFRL-IF-RS-TR-2003-254, Cigital Labs, 2003.

[23] P. Slowikowski and K. Zielinski. Comparison study of aspect-oriented and container managed security. In *Proceedings of the ECCOP Workshop on Analysis of Aspect-Oriented Software*, 2003.

[24] J. Yoder and J. Barcalow. Architectural patterns for enabling application security. In *Proceedings of the 4th Annual Conference on the Pattern Languages of Programs (PLoP97)*, 1997.