Incorporating OAuth Protocol into Existing Information Systems

Utharn Buranasaksee*
Rajamangala University of Technology Suvarnabhumi, Phranakhon Si Ayutthaya, Thailand.

* Corresponding author. Email: utharn.b@rmutsb.ac.th
Manuscript submitted January 7, 2016; accepted April 1, 2016.
doi:10.17706/jsw.11.6.615-622

Abstract: Traditionally, when the user wants to share his resource on one application to another application, the user needs to give his credential to another application that causes the privacy issues. Then OAuth protocol was introduced to solve the problem without providing the user's credential. The protocol was also designed to support mobile, desktop, and web applications. This makes OAuth protocol an essential functionality for a newly developed project. Therefore, the need of migrating the current data repositories to support authorization as an OAuth server gains more attention. However, there have been many software libraries publicly available to download; all of them focus on processing and constructing the messages that comply with the OAuth standard. None of them provides the channel that a new OAuth server could integrate the existing resources with their business logics to the service. This paper pointed the issues and proposed the object-oriented class design that can solve the problems. Then the prototype was implemented and shown that it supports reusability of the existing business logics.

Key words: Incorporate, oauth, design, business logic, existing information system.

1. Introduction

With the evolution of Web 2.0, many web applications on the Internet start allowing the user to grant his information to another web application in a different domain to access information on behalf of the user. For example, a student needs to submit TOEFL score from ETS to the university. However, it is not possible for the student to submit the score himself due to lacking for authenticity proof. Instead, the university rather wants to obtain the score at the ETS itself. To achieve that, a student could buy an electronic access of his score from ETS and share this access to the university. Using traditional methods, the student could give his account credential to the university so that the university staff can access to his account. However, this creates the privacy issues since the university could gain full access control of the student account. Then there OAuth protocol was introduced to address this problem by allowing a student to grant his access to the university without giving his credential.

OAuth protocol [1], [2] is an open authorization protocol that enables the user to authorize one application to access the user's resource on behalf of the user. The main application of OAuth-based protocol is sharing the protected resource on behalf of the user. This also includes the user identity information, which will be used for authentication functionality. OpenID Connect [3] provides the user identify information on top of the OAuth 2.0 protocol. The protocol is designed to be compatible with OAuth 2.0 protocol and to support claims-based authentication. A claims-based authentication is a novel approach to acquiring the user identity information they need on the internet. A claim is a statement that one subject makes about itself such as name, email address, etc. The concept of issuer provides the way to certify that the claims are authentic. In another word, instead of having the OAuth client fetching information from the

OAuth server, the OAuth server could also provide authentic information as the information directly comes from the OAuth server.

Since the first version of the OAuth protocol [1] was originally designed for the web applications, the current version of OAuth protocol [2] itself works on HTTPS protocol, one of the most popular protocol on the Internet. Therefore, the protocol is designed to support the web, mobile, and desktop applications. This is very useful when, in an organization, many projects are developing over time and these applications performs account information synchronization and enforcement through some custom proprietary mechanisms on traditional database like direct access to the database, LDAP, etc. Since these protocols are not designed for mobile and Internet applications, implementation between the applications over Internet would be difficult as it requires additional firewall rules for accessing or synchronizing passwords. Therefore, OAuth gains more popularity over time and many newly developed application are designed to support OAuth and OpenID connect as the client. Therefore, of converting traditional authentication repositories to supports OAuth protocol for OAuth server gains more attention.

Migrating to OAuth based protocols is a simple process. In authorization process, both OAuth server and OAuth client just need to be able to produce the message that complies with the standard and process the message sent from another party. However, in the resource accessing process, a different organization would have different information requirements and the account information including the user data which will be authorized throughout the process could be various from organization to organization. Publishing this information requires custom implementation, as there is no detail in the protocol specification. As the result, the message structure can be vary depending on the returned data type. In some application context, the existing business logics can be complex, rewriting data access layer or business logics are not possible. Though there have been many public libraries for OAuth protocol that have been implemented in various programming languages such as Java, PHP, .NET framework, Ruby, etc.. These software libraries and their examples are focusing on how to produce and process the messages sent to the OAuth client, the OAuth server, and the user. None of them focuses on integrating the OAuth server to the existing resource server. Therefore, this paper makes the following contribution. First, it defines the problem of the implementation where rewriting authorization of the existing resource could be a complicated task. Second, it uses the design pattern techniques to solve the problems. Finally, the advantages of the proposed design are discussed.

The remainder of the paper is structured as follows. In Section 2, the problem statement and environment are defined. Then the existing software libraries and research works related to OAuth design are presented and analyzed in Section 3. Section 4 demonstrates the design of the proposed system in different design pattern categories. Then the advantages and the implementation are discussed in Section 5. Finally, the conclusions are drawn in Section 6.

2. Problem Statement

This work focuses on an OAuth 2.0 based system that needs to implement an OAuth server role as a newly developed project while providing existing resource access through endpoints. Each resource type is accessed via different provided endpoints. The OAuth server contains the existing database structure with its data. Accessing or modifying the resources requires complex business logic that interacts with another module. The server wants to reuse the existing components from the existing project. Therefore, the server requires no change and no rewrite the data access layer or business layer of the resource.

Therefore, this paper proposed the class design by using design pattern techniques to support the following functionalities. First is to promote reusability of the existing business objects to support the OAuth process. Second is to support the separation of concern design of the process in message procession

and message production to provide a base infrastructure of the library. Finally, the design that supports accessing different authorized resources datatype is proposed.

3. Literature Reviews

In this section, the concept of design pattern is explained in the first topic. Then the processes of the OAuth based protocol are elaborated, and the existing works are discussed in the next topic.

3.1. Design Pattern

Though the concept of design pattern was originated by Gamma *et al*. [4], there have been many design patterns proposed afterward to solve new problems such as domain-driven [5], patterns in enterprise architecture [6]. In software engineering, the design pattern is a concept of making a reusable solution to the common problem within a given context. A reusability is one of the software properties of which the code can be used in the different application with minimal change. Therefore, that new application partially inherits the code attributes and functionality when the code is reused in a new application. The patterns are usually formalized the best solution that the programmer can solve the problem when designing the system. In one software solution, there could be many problems which design pattern can be used to apply differently. The combination of the best design patterns typically results in the best practices in a software solution.

In object-oriented design pattern, classes and objects are viewed as they interact each other. There are three types of design patterns. First, creational design patterns are the design patterns that deal with how the objects are created in a suitable situation. Second, structural design patterns are the design patterns that define ways to compose or obtain new abilities or functionalities. Third, behavioral design patterns are the design patterns that concern with communication between objects. In this paper, the discussed problems are solved using three design pattern areas altogether.

3.2. OAuth Based Protocol

In OAuth, there are three parties, which are OAuth Server, OAuth Client, and the user. When the user authorizes the OAuth Client, the user clicks on the link constructed by the OAuth Client. Then the user is redirected to the OAuth Server with the requested authorization information from the OAuth Client. At the OAuth server, the user is asked to identify himself and authorize the request. After authorization is done, the OAuth server redirects the user back to the OAuth Client with an access token. Then OAuth Client can use this access token to access the resource on the behalf afterward.

When implementing an OAuth server, five data types need to be stored in the database. First, OAuth client is the applications that have registered and contains connection information such as secret key, authorized endpoints, type of application, etc. Second, OAuth scope refers to a set of strings that represent the permissions that OAuth client would like to access. The permissions are usually associated with a set of fields or attributes that the OAuth client can access from the OAuth server. Since the connection typically made over the Internet, multiple scopes are allowed in the same request in order to reduce roundtrip time. Third, OAuth token is the information that will be used to represent authorization between OAuth client and OAuth server. There are two types of token. First, an access token is used by the OAuth client to access the authorized resources. Since an access token will frequently be used over the Internet, it is designed to be short-lived. If an access token is expired, a new token needs to be requested. The other type of token is a refresh token. A refresh token is a token that will be used to request to a new access token without authorization again. Since a refresh token is not frequently used, it is designed to be long-lived. Fourth, unlike typical user database that contains username and password, OAuth user is the user database that has linked to authorized OAuth clients to access their resource. Typically, OAuth user should be the same as the

existing user database, but it contains authorization attributes in addition to the attributes as in the existing database. Finally, the authorized resource is typically an existing resource in the database that will be requested by the OAuth client through a given endpoint.

Once the OAuth client is authorized by the user, the OAuth server issues an access token so that the OAuth client can use to access the authorized information. Therefore, each OAuth client contains multiple access tokens for its user and each authorization is associated with multiple scopes. As the result, the returned structure is determined at runtime. For example, when requesting user information from an userinfo endpoint using openid scope, the OAuth server would return the serialized object with the default fields as id, name, firstname, and lastname. However, if an additional scope called email were requested, the OAuth server would return email field in addition to the default field.

Since OAuth is the most popular authorization protocol on the Internet, there have been many software libraries publicly available to download and implement [7]-[12]. However, some of the software libraries [8]-[11] use a bridge design pattern in order to support many database management system providers but they support only a new database of which the structures are created according to the libraries. Some [12] uses a repository design pattern in order to support different types of data mapping layer such as entity framework and asp.net identity. In [12], the existing data can be used in the OAuth server, but these are usable only when the existing applications use asp.net identity framework. In the research area, [13] focuses on improving flexibility and extensibility by which if the content and structure of the messages are changed, these changes are still making the final output of the message compatible with the previous version of the protocol. [14] focuses on extending OAuth-protocol based message to support more features. [15]-[17] proposed the reusable design which can be applied in the OAuth context but the work focuses on the OAuth client party. None of them focuses on integrating the OAuth server to the existing resource server as the OAuth client, scope, token, and OAuth user are highly coupled, making changes to one of them would require updating for the rest of the system.

Therefore, the purposes of the proposed design are as follows. First, it reduces coupling of the system. Then a novel design of processing and producing the OAuth message for each request is proposed to separate the work on message request and response. With the proposed creational design, the implementation of the OAuth server could be extended to support specific resource type. Finally, the structural design is proposed to support reusability of the existing business logics.

4. Proposed Design

As the OAuth database designs in all the existing works were designed to support OAuth only features with a newly created database, the proposed design would be separated into three steps according to popular design pattern categories. Then the advantages of each step will be discussed.

4.1. Behavioral Design

According to the process of the OAuth authorization, each step involves in checking process of client, scope, and token. This information is processed together in order to process the request and produce the OAuth response back to the OAuth client. As the classes are highly coupled, changing the code in one of these classes would result in changing the logic in another class. Therefore, the Mediator design pattern is proposed to separate the classes and reduce the number of parameter passing among the classes. The Mediator is the concept of the pattern that encapsulates how the objects interact by keeping the objects from direct calling. Therefore, reducing the number of lines of code changes in another class if one of them changes.

In OAuth protocol context, the endpoint running on the web server would accept the request from the

OAuth client and produce the OAuth response back to the OAuth client. Therefore, the proposed design introduces the superclass called *OAuthFormatter* to handle OAuth request and OAuth response from HTTP request and HTTP response. Then the *OAuthParser* and *OAuthResponder* inherit *OAuthFormatter* in order to implement discrete functionality in request and response process accordingly.

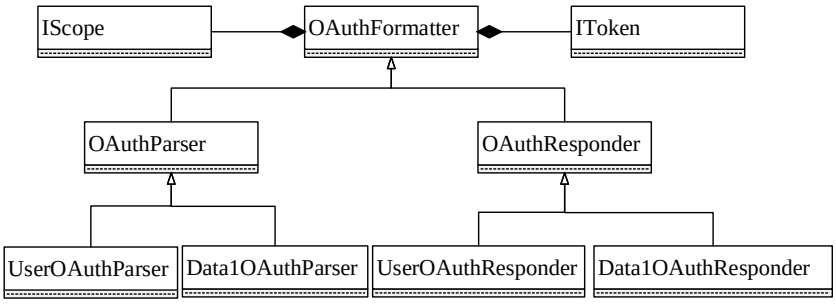

Fig. 1. UML diagram of *OauthFormatter* class.

For each added endpoint, more discrete functionalities could be implemented by inheriting both *OAuthParser* and *OAuthResponder*. For example, as shown in Fig. 1, a class *UserOAuthParser* and *UserOAuthResponder* inherits *OAuthParser* and *OAuthResponder* respectively in order to handle OAuth request and OAuth response for the user class. In addition, the *OAuthFormatter* processes the scope and the token by referencing *IScope* and *IToken* interfaces.

4.2. Creational Design

In the OAuth protocol process, once the user granted the authorization request at the authorization endpoint, the OAuth client would use the access token to request the resource at different endpoint afterward. Since the process after the authorization would follow the same pattern except that the resource type returned to the OAuth client, Abstract factory and Factory method are used together in the proposed design in order to provide an infrastructure of the resource procession. In Abstract factory, there are two types of class. One is a factory class that defines how the objects are created. A Factory class produces the other class called product class. Each factory would produce its own product only. The discrete factory could be implemented by inherited factory superclass and the discrete product created from the discrete factory would need to inherit the same hierarchy of the product superclass. In addition, in order to provide a more standardized way to create the product class. The factory class also uses factory method pattern. The factory method pattern makes all the subclasses of the same factory class implements the same method of product creation.

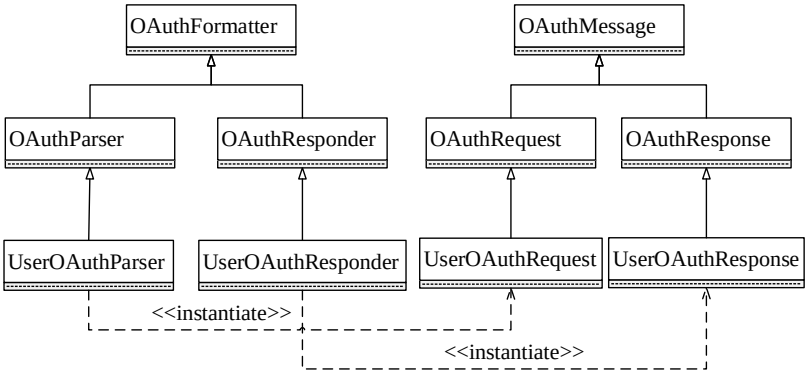

Fig. 2. UML diagram of *OAuthFormatter* and *OAuthMessage*.

In OAuth protocol context, the *OAuthFormatter* produces *OAuthMessage*. In *OAuthMessage*, there are two subclasses, which are *OAuthRequest* and *OAuthResponse*. The *OAuthParser* handles *OAuthRequest* and the *OAuthResponder* produces *OAuthResponse* accordingly.

For each added endpoint, the more discrete data structure could be implemented by inheriting both *OAuthRequest* and *OAuthresponse*. For example, as shown in Fig. 2, a class *UserOAuthRequest* and *UserOAuthResponse* are the product of the *UserOAuthParser* and *UserOAuthResponder* respectively. Furthermore, the methods used in subclasses of *OAuthParser* and *OAuthResponder* would implement the same such as readRequest(), createResponse(), etc.

4.3. Structural Design

As from the requirements in this paper, the proposed structural design of the existing resource that will be published as an authorized resource through an endpoint. To achieve this requirement, an adapter pattern is used in this scenario. Adapter design pattern is typically used when a new system wants to reuse the existing business objects without modifying their business logics or data access layer as they are already implemented, or their creation process could be complicated.

Note that, in the design of scope and token, a bridge design pattern is used to separate the structure of them from the implementation using *IScopeStore* and *ITokenStore*, we leave the discussions of scope and token out of the paper due to the space limitation. In addition, some of the existing works also use this pattern to manage them.

In OAuth protocol context, the *IDataAdapter* interface has proposed to fulfill the above requirement. *IDataAdapter* interface provides a basic create, retrieve, update, and delete operations (CRUD operations) from the existing business logics. Therefore, to reuse the existing business objects, a new class that implements *IDataAdapter* and accepts the existing business object as a constructor can be defined. The code implemented in *IDataAdapter* class will call the function in its adaptee class. To attach *IDataAdapter* into the proposed design, the object is contained in the *OAuthMessage*, which means that the OAuth client can put the object to update the data on the OAuth server using *OAuthRequest* and the OAuth client can retrieve the object from the OAuth server via *OAuthResponse*.

For each added endpoint, more discrete data adapter could be implemented by implementing *IDataAdapter* and uses the discrete data adapter as an object in *OAuthMessage*. For example, as shown in Fig. 3, a class *UserDataAdapter* that implements *IDataAdapter* that calls existing business logics in *UserDataAdaptee*. The *UserDataAdapter* accepts the existing business object User is contained in *UserOAuthRequest* and *UserOAuthResponse*. The OAuth client can embed the User object in the request and update the data in the OAuth server using *UserOAuthRequest*. In addition, the OAuth server can serialize the User object through *UserDataAdapter* and embed in *UserOAuthResponse*.

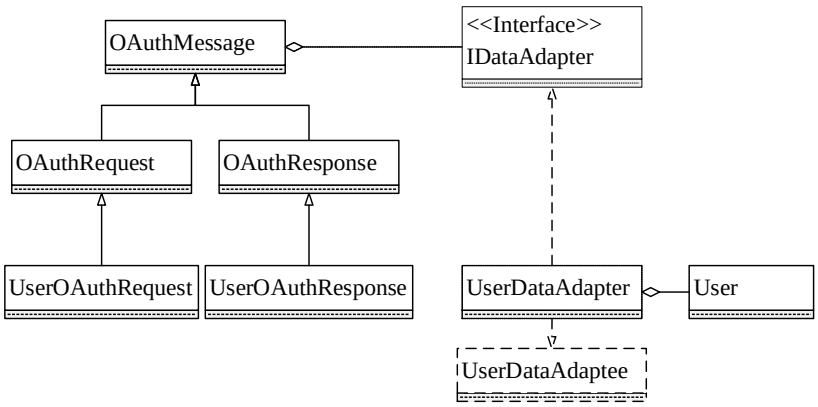

Fig. 3. UML diagram of between *IDataAdapter* and *OauthMessage*.

5. Advantages and Implementation

The proposed design takes the benefits of the design patterns to improve the reusability of the existing business logics in the organization. In this section, the comparison of the proposed design and the design of the existing studies were compared in Table 1. Since some of the software libraries just focus on the OAuth message production and parsing, the code we downloaded uses hard coding techniques just to illustrate how the code works. Therefore, [7]-[11] does not apply the design pattern techniques in some categories. Except the proposed design, only [9], [12] apply the design pattern techniques in all three categories. However, the author still does not leave any interfaces or abstract classes to connect to the existing resource. Finally, [11] introduces data API interface for the existing resource, but modifying the related classes such as scope is difficult since the library does not use a behavioral design pattern technique.

Our prototype is implemented using Visual Basic.NET. However, our proposed design could also be used in many popular languages such as Java, PHP, or ruby.

Table 1. Comparison among the Proposed Design and the Existing Libraries

Software Libraries	Coupling	Behavioral patterns	Creational patterns	Structural patterns	Reusability of existing business logics
Proposed design	Low	Mediator	Abstract factory & Factory method	Adapter & Bridge	Supported
Apache Oltu [7]	High	-	Abstract factory & Builder	-	Not Supported
OAuth2 Server in PHP [8]	High	-	-	Bridge	Not Supported
OAuth for Spring Security [9]	High	Chain of responsibility	Factory method	Bridge	Not Supported
PHP OAuth 2.0 Server [10]	High	Chain of responsibility	-	Bridge	Not Supported
DotNetOpenAuth [11]	High	-	-	Adapter & Bridge	Not Supported
IdentityServer3 [12]	Low	Mediator	Abstract factory	Repository & Bridge	Not Supported

6. Conclusion

We introduced the issues in implementing the OAuth server role where the software libraries available on the official website have not addressed. Then we proposed the class design that addressed the problem.

The proposed class design was conceived with the object-oriented programming language, the most popular programming approach. The class design was generic enough to implement in any other object-oriented languages. Then we elaborate the problem and explain how the proposed class design addresses the problem. Finally, the proposed class design was evaluated with the existing software libraries. We showed that our proposed class design was able to solve the issues.

References

[1] Hammer-Lahav, E. (2010). The OAuth 1.0 protocol.

[2] Hardt, D. (2012). The OAuth 2.0 authorization framework.

[3] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., & Mortimore, C. (2014). OpenID connect core 1.0. *The OpenID Foundation*, S3.

[4] Vlissides, J., Helm, R., Johnson, R., & Gamma, E. (1995). *Design patterns: Elements of reusable object-oriented software*. Reading: Addison-Wesley, *49(120),* 11.

[5] Evans, E. (2004). *Domain-driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley Professional.

[6] Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc.

[7] The Apache Software Foundation. Apache Oltu. Retrieved December 1, 2015, from http://oltu.apache.org/

[8] Shaffer B. OAuth2 Server in php. Retrieved December 1, 2015, from https://github.com/bshaffer/oauth2-server-php

[9] Pivotal Software Inc. OAuth for Spring Security. Retrieved December 1, 2015, from http://projects.spring.io/spring-security-oauth/docs/Home.html

[10] Bilbie A. PHP OAuth 2.0 Server. Retrieved December 1, 2015 from http://oauth2.thephpleague.com/

[11] Aarnott A., Christiansen D. DotNetOpenAuth. Retrieved December 1, 2015, from http://dotnetopenauth.net/

[12] Dot net foundation. IdentityServer3. Retrieved December 1, 2015, from https://github.com/IdentityServer/IdentityServer3

[13] Buranasaksee, U., Porkaew, K., & Supasitthimethee, U. (2014). Ticket model: A generalised model for internet-based three-party authorisation systems. *International Journal of Internet Protocol Technology, 8(4),* 159-168.

[14] Buranasaksee, U., Porkaew, K., & Supasitthimethee, U. (2014, February). AccAuth: Accounting system for OAuth protocol. *Proceeding of 2014 Fifth International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)*, (pp. 8-13). IEEE.

[15] Hashimoto, R., Ueno, N., & Shimomura, M. (2009, November). A design of usable and secure access-control APIs for mashup applications. *Proceedings of the 5th ACM workshop on Digital identity management* (pp. 31-34). ACM.

[16] Aghaee, S., Nowak, M., & Pautasso, C. (2012, June). Reusable decision space for mashup tool design. *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (pp. 211-220). ACM.

[17] Rodríguez, C., Chowdhury, S. R., Daniel, F., Nezhad, H. R. M., & Casati, F. (2014). Assisted mashup development: On the discovery and recommendation of mashup composition knowledge. *Proceeding of Web Services Foundations* (pp. 683-708). Springer New York.

Utharn Buranasaksee received his PhD in Computer Science from King Mongkut's University of Technology Thonburi, thd master in information technology from King Mongkut's University of Technology Thonburi, Bangkok and BCA degree from Christ College, Bangalore University. He is a lecture in computer science field at Rajamangala University of Technology Suvarnabhumi. His research interests include keyword search, spatial database, web technology and internet protocols design.