

# **An Implementation of a Middleware for Client and Server Binding and Load Balancing**

Dawei Liu

Department of Computer Science

University of Kentucky

Lexington, KY 40506

Advisory Committee

Dr. Dakshnamoorthy (Mani) Manivannan

Dr. Zongming Fei

Dr. Tingting Yu

Date: 04/17/2016

## Table of Contents

1. Introduction.....	4
1.1 Background.....	4
1.2 Project Scope .....	4
2. Requirements Definition.....	5
2.1 Objectives .....	5
2.2 Requirements .....	5
2.2.1 Assumptions.....	6
2.2.2 Connection Binding .....	6
2.2.3 Load Balancing .....	6
2.2.4 High Level APIs .....	6
2.2.5 Communication Security.....	7
3. System Architecture.....	7
3.1 Overview.....	7
3.2 Modules.....	8
3.2.1 Basic Layers.....	8
3.2.2 Stub Layer.....	9
3.2.3 Middle Layer.....	9
3.2.4 Hub.....	10
4. Functional Design .....	11
4.1 System Requirements.....	11
4.1.1 Hardware Requirements.....	11
4.1.2 Software Requirements .....	11
4.2 Design .....	12
4.2.1 Hub.....	12
4.2.2 Server Middle Layer .....	13

---

4.2.3	Client Middle Layer .....	15
5.	Implementation .....	15
5.1	Implementation Challenges.....	15
5.2	Module Implementation.....	16
5.2.1	UtilsRuntime .....	16
5.2.2	XMLParser.....	18
5.2.3	ESBCommonLib.....	19
5.2.4	ESBWebService.....	19
5.2.5	ESBDataSerializer .....	19
5.2.6	ESBMidService.....	20
5.2.7	ESBMidConn.....	21
5.2.8	ESBHubServiceLib.....	21
5.2.9	HubService.....	22
6.	System Test.....	22
6.1	Sample Development.....	22
6.1.1	Server .....	22
6.1.2	Client.....	22
6.2	Sample Test.....	23
6.2.1	Scenario 1. Run the Server while the Hub is not started/running. ....	23
6.2.2	Scenario 2. Run the Server when the Hub has already been started. ....	23
6.2.3	Scenario 3. Stop the Hub after the Server has already been registered on the Hub. ....	24
6.2.4	Scenario 4. Start a Client while the Hub has not been started. ....	24
6.2.5	Scenario 5. Start a Client while the Server has not been started. ....	24
6.2.6	Scenario 6. Start a Client when the Hub and the Server have already been started. ....	24
6.2.7	Scenario 7. Query database from the terminal of the client.....	25

---

6.2.8 Scenario 8. Query database from the terminals of two clients.....	25
6.2.9 Scenario 9. A Client connects to the Server whose connection pool is full.....	26
6.2.10 Scenario 10. A Client connect to the Server whose connection pool is not full. ....	26
6.2.11 Scenario 11. Load Balancing .....	27
7 Lines of Code .....	28
8. Future Works .....	29
8.1 Optimizing the Implementation of Load Balance .....	29
8.2 Using Lock-Free Hash Map Instead of std::map .....	29
8.3 Environmentally Independent XML Parser .....	29
8.4 Cross Platforms .....	29
8.5 APIs for Cross Languages.....	29
10. References .....	30

---

## 1. Introduction

### 1.1 Background

In conventional Client/Server (C/S) architecture, a server need to respond to connections from large number of clients. It would make severe overload on the server, and slow down server's processing capacities. The slow processing would degrade performance of network transmission between the clients and server. If the surge of connection requests exceeds the capacity of the server or the network, the server or network devices may crash.

A classic solution is to set a limitation on the maximum number of connections for the server. Once the number of concurrent connections exceed the limitation, the server would reject future connection until the number of connections get lower than the limit. This solution is widely used for small service.

However, if the number of connection required to be handled by the server exceeds the capacity of a conventional server, this is not a solution. Therefore, I suggest forming a service group formed with several servers and a Hub. Each server would provide similar or different service, and the Hub would distribute the load among the servers.

The system would has following functions:

- **Dynamic connection binding.** The *Hub* would offer centralized service management. All the servers in a group would register themselves on the *Hub*. Thus, a client can query different servers available from one end-point, the *Hub*.
- **Load Balancing.** Servers with the same functionalities can be registered on the same *Hub*. The Hub would order the list of the registered servers by their load status automatically. Therefore, the Hub knows the idlest servers in the lists of different type of services. Thus, when a client sends a server-connection request to the Hub with a service ID, the Hub would respond the client with the idlest service from the service list of registered servers.
- **High level Application Program Interfaces (APIs) for developers.** Developer can use this set of APIs to design C/S structure applications with the benefit of the system, including managed load control and one-end-point connection binding. Programmer would never care with the detail network protocol and communication with a Hub, since these would be hidden by the APIs.

### 1.2 Project Scope

The *Hub* of this system would be the center point that clients connect to get a service. It is visible for deployment. Apart from the Hub, two other important parts are Client Middle Layer and Service Client

---

Layer. Developers can use these two parts to program client/server structure applications with the functionality of the system. They are only visible to developers.

Hub *does*:

- Allow servers register their services with it.
- Allow clients to retrieve valid services by type IDs.
- Perform client-server binding based on the load of the server status.
- Ensure the connection validations of registered services by receiving heartbeat from the services.

Hub *does not*:

- Allow developer-defined communication protocol between the client and the Hub.
- Allow developer-defined communication protocol between the service and the Hub.

Client Middle Layer *does*:

- Retrieve service from a Hub and then connect to the server providing the service.
- Allow developer-defined communication protocol between client and server.

Client Middle Layer *does not*:

- Allow developer-defined communication protocol between the client and the Hub.
- Connect to the service directly without permission from the Hub.
- Keep the validation of connection by sending Heartbeat messages to the server.

Service Middle Layer *does*:

- Allow developer-defined communication protocol between the client and the service.
- Update its load status on the Hub when necessary.
- Keep the validation of registration by sending Heartbeat messages to the Hub.

Service Middle Layer *does not*:

- Allow developer-defined communication protocol between the service and the Hub.
- Keep the validation of connection by sending Heartbeat messages to the client(s).

## 2. Requirements Definition

### 2.1 Objectives

Build a system that supports dynamic connection binding and load balancing. Meanwhile a set of APIs would be introduced to develop client and service applications.

### 2.2 Requirements

---

### 2.2.1 Assumptions

- 1) All services are required to be registered on a central location, called Hub. Different functionalities of services can be identified by predetermined IDs.
- 2) A client cannot connect to a server if it has not retrieved a valid session key from the Hub with which the server registered. From the Hub, which manages different types of services, the client can get a URL of specified server by a predetermined ID. Thus, the Hub would be responsible for the **connection binding** between a client and a server.
- 3) If there were more than one servers with the same type of service registered on a Hub, the Hub would manage the load status of these servers. The hub would ensure the every server would have the similar load rate (in percentage). This function is called **load balancing**.

### 2.2.2 Connection Binding

- 1) A client can retrieve the connection binding from a Hub.
- 2) All services are required to be registered on the Hub.
- 3) Each type of service would be specified with a unique ID. The Hub would use the IDs to identify different services. In addition, a client would use an ID to retrieve the proper service from Hub, and Hub would send the client a token with a URL of the service as a valid connecting permission for client. Moreover, the token contains the session key that is required to connect to the service. The server would get the token from the Hub to validate the connection from the client.

### 2.2.3 Load Balancing

- 1) Server which offer the same functionalities are considered to be of the same type. In this case, the servers would be registered with the same ID. Therefore, one ID could be hold by several servers.
- 2) Each server would inform its load capability to the Hub periodically.
- 3) Once a client sends a connection request for a service to the Hub with an ID, the Hub would find out the proper service which service is the idlest in the list of services with the same ID. Then the Hub would send the client with the IP address of the server offering the service and the valid session key so that the client can connect to the server.

### 2.2.4 High Level APIs

---

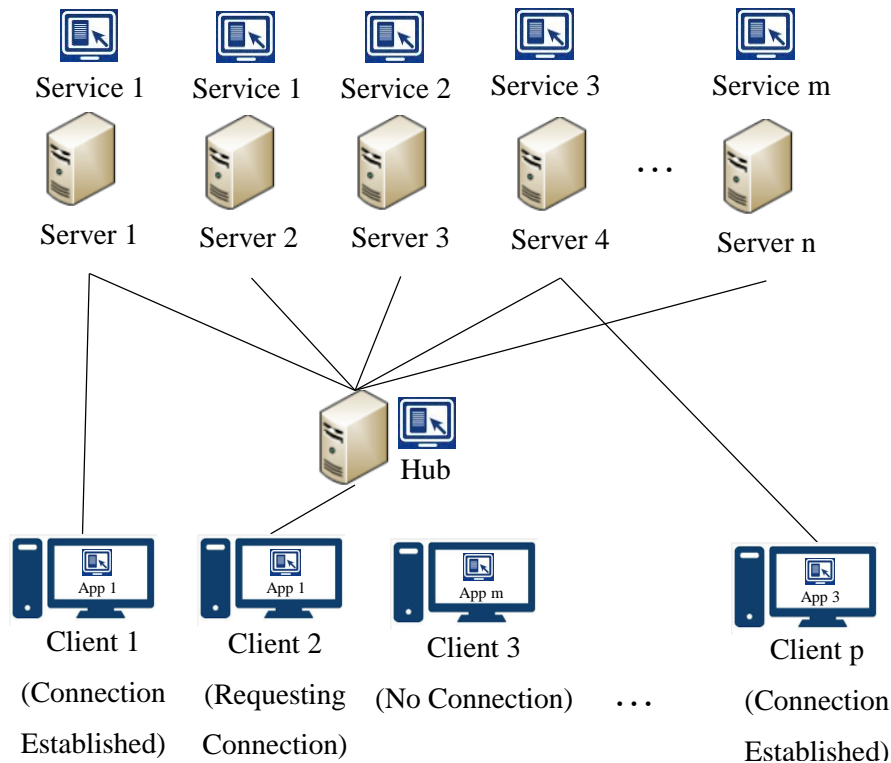
Despite of the *Hub*, all other services and client terminates would be provided by third parties. Therefore, the system provide high level APIs for developers to help them design their products using the functions of the system. Developers would be beyond the scope of detail messages between the *Hub* and their services designing.

### 2.2.5 Communication Security

Encryption of message may be required in communication. Authentication management is optional. To enable secured communication, the Client-Requiring-Server Authentication should be specified by developer. In this scenario, server should keep its private key and public key, which are signed by a CA (Certificate Authority), while clients would be required to keep the CA's public key.

## 3. System Architecture

### 3.1 Overview



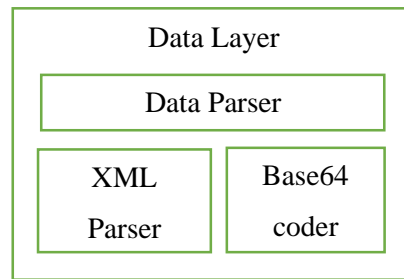
**Figure 1**



As **Figure 1** shows, each server would have a service, that would be registered with the Hub. The Hub would group the servers into different lists by their services' IDs and manage the load of each services. Meanwhile each client would have an App (application), that its ID would be same to the ID of the corresponding service. When a client needs to connect to a server, as *Client 2* in **Figure 1**, it would firstly request Hub with the Service/App ID for a URL of a server and a valid session token to connect to the server. The Hub would find out a proper service and its server's URL based on overall load of the services that has already registered on it, and respond the client with the information it needs. With the information the Hub responded, the client would connect to the proper server directly, as shown with *Client 1* and *Client p* in **Figure 1**.

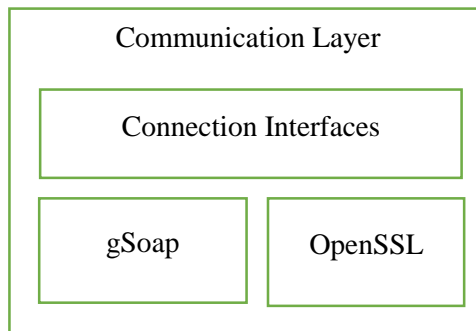
### 3.2 Modules

#### 3.2.1 Basic Layers



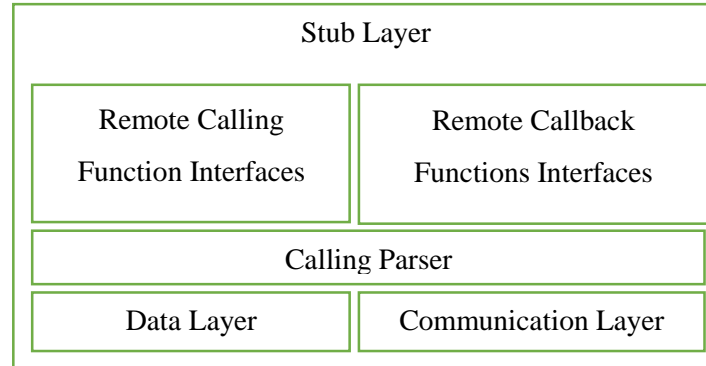
**Figure 2**

Data Layer Library is statically shared in the client, server and hub. Using this library, binary data can be easily translated into string, and a string can be converted back to the original binary data. Data Parser would provide upper layer with its interfaces for development; XML Parser would be used to format the string data; Base64 coder, which uses binary-to-text encoding schemes that represent binary data in an ASCII string format by translating it into a radix-64 representation [1], would be used to encode linear binary data to a string and decode a string to linear binary data.

**Figure 3**

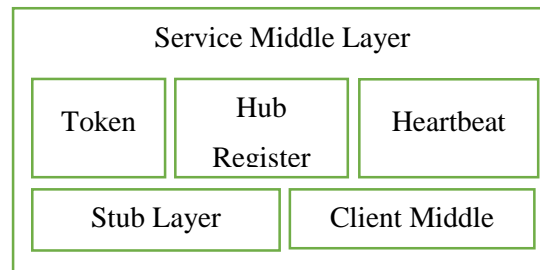
Communication Layer Library is dynamically shared in the client, server and hub. It abstracts the communication between any pair of server, hub and client. gSoap would be used for the low level network communication, and OpenSSL would be used to ensure secure communication.

### 3.2.2 Stub Layer

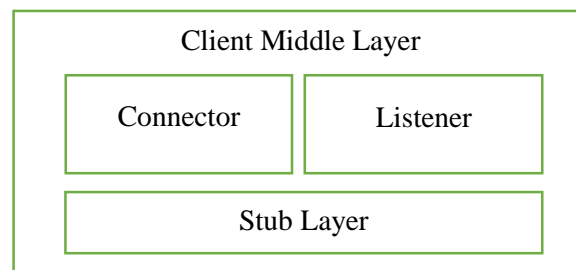
**Figure 4**

Stub Layer is used to translate a function call to a network stream that would be transmitted to another node. Also, when a node received this calling stream, the node would use the layer to transmit the string to function calling event. After the function is called, the node would use the layer to transform the return value to a network stream that would send back to the caller node.

### 3.2.3 Middle Layer

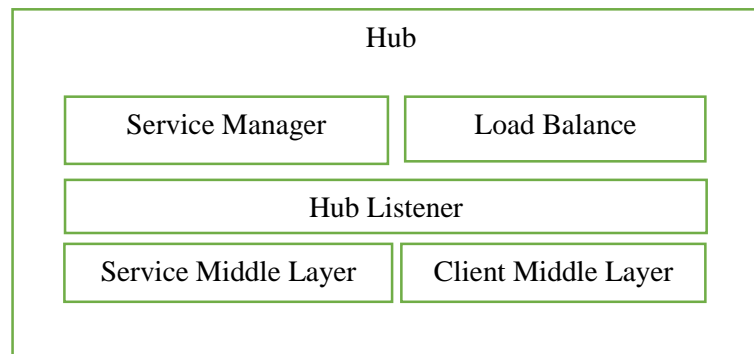
**Figure 5**

Service Middle Layer is a series of interfaces that abstract the communication between the Servers, Hub, and Clients. Service would use this middle layer to register itself on the Hub, and report its load status automatically.

**Figure 6**

Client Software can use Client Middle Layer to simplify the connection with Hub and Service.

### 3.2.4 Hub

**Figure 7**

The main work for Hub is to dispatch the connections from clients to proper services based on overall load status. It would manage the registration of services and the connection requests from client. Also the Hub can use Token to control the authorization of connections.

## 4. Functional Design

### 4.1 System Requirements

Absence of required features would cause unexpected problems and even system failures. Also the hardware requirement would directly affect the efficiency of the system. The guideline of environments for the system are listed as following.

#### 4.1.1 Hardware Requirements

Scenario	Required processor	Recommended processor	Required RAM	Recommended RAM
Client	Intel® Atom™	Intel® Core™ i3	128 MB	256 MB or higher
Server/Hub	Intel® Core™ i3	Intel® Core™ i5	256 MB	512 MB or higher
Development	Intel® Core™ i3	Intel® Core™ i7	1024 MB	4096 MB or higher

#### 4.1.2 Software Requirements

##### 4.1.2.1 Operating System

---

Scenario	Operating system
Client	Microsoft® Windows® 7 or higher
Server/Hub	Microsoft® Windows® Server 2008 or higher
Development	Microsoft® Windows® 7 Professional or higher

#### 4.1.2.2 Additional Software

Scenario	Software
Client	Microsoft® Internet Explorer 7.0 or higher
Server/Hub	Microsoft® Internet Explorer 7.0 or higher
Development	Microsoft® Internet Explorer 7.0 or higher Microsoft® Visual Studio 2015 or higher

## 4.2 Design

This section would show the design of three important layers: Hub, Service Middle Layer, and Client Middle Layer, since the interaction between these three layers consist all the functionalities of the system. While lower level under these layers would focus on the compatibility for operating systems, network protocols and hardware, and higher level beyond these layers would focus on the functionalities of applications and UI/UX.

### 4.2.1 Hub

#### 4.2.1.1 Initialize

When hub is initialized, it would listen on the network via predetermined port to wait for the connections from clients and service's registration.

#### 4.2.1.2 Terminate

---

- 1) Stop listening on the network, and send token revoked messages to all the service. All resources allocated by the Hub should be cleared.

- 2) Stop heart beating.

Prerequisite: Initialized

#### **4.2.1.3 Receive a service registration message**

- 1) Register the service into the service table.

- 2) Respond with success message.

Prerequisite: Initialized

#### **4.2.1.4 Receive a service unregistration message**

Unregister the service

Prerequisite: Initialized; the service required to be unregistered has already signed in the Hub.

#### **4.2.1.5 Get a connection from a client**

- 1) Get connection information from the network stream, which contains the information for requested service.

- 2) Find out the proper service by services' load status.

- 3) Generate a token.

- 4) Send the token to the proper service.

- 5) Respond the client with the token and the IP address of the server

- 6) Start counting the heartbeat of client.

Prerequisite: Initialized; received a connection from a client.

#### **4.2.1.6 Heartbeat message**

Send Heartbeat message to all the registered services every predetermined period.

Prerequisite: Initialized.

### **4.2.2 Server Middle Layer**

#### **4.2.2.1 Initialize**

- 1) Listens to the network via a predetermined port.

- 2) Register itself with the *Hub*.

---

- 3) Startup session management.
- 4) Wait for messages from the Hub and clients.

#### **4.2.2.2 Terminate**

- 1) Stop session management.
- 2) Stop listening.
- 3) Unregister itself with the Hub
- 4) Stop sending heartbeat messages.

Prerequisite: Initialized.

#### **4.2.2.3 Register itself with a Hub**

- 1) Send registration message to the Hub.
- 2) Wait for response from the Hub for a timeout period.
- 3) After timeout period, if acknowledgement is not received from the Hub, return a failure value.
- 4) Otherwise, return a success value.

Prerequisite: Initialized.

#### **4.2.2.4 Unregister itself with the Hub**

- 1) Send unregister message to the Hub.
- 2) Wait for response from the Hub for a predetermined period.
- 3) If timeout expires, then we assume the Hub has already known the service stopped.
- 5) Send token expired message to all the clients.
- 6) Stop sending heartbeat messages.

Prerequisite: Initialized and registered on the Hub.

#### **4.2.2.5 Receive a token message**

Save token into a map table.

Prerequisite: Initialized.

#### **4.2.2.6 Receive a request for connection from a client**

- 1) Processes information from the stream, and then it gets token and operation message.
  - 2) Determines if the token is already in the token table.
-

- 3) If the token is not in the token table, then send refuse message to the client.
- 4) Otherwise, processes the operation message and then send a reply to the client with the results.

Prerequisite: Initialized.

#### **4.2.2.7 Receive a token revoked message**

Remove the token from the token table.

Prerequisite: Initialized; the token already exists in the token table.

### **4.2.3 Client Middle Layer**

#### **4.2.3.1 Connect**

- 1) Connect to the hub.
- 2) Wait for the reply for a predetermined time.
- 3) Get the IP address of the remote host and a token.
- 4) Set status as Connected.
- 5) Start heartbeat.

Prerequisite: Unconnected.

#### **4.2.3.2 Disconnect**

- 1) Send a token revoked message to the remote host.
- 2) Set status as disconnected.

Prerequisite: connected

#### **4.2.3.3 Receive a token expired message**

Set statue as disconnected.

Prerequisite: Connected

## **5. Implementation**

### **5.1 Implementation Challenges**

- 1) Keeping high cohesion and loose coupling. Cohesion in software engineering is the degree to which the elements of a certain module belong together. Thus, it is a measure of how strongly related each piece of functionality expressed by the source code of a software module is. Coupling in simple words, is how much one component knows about the inner workings or inner elements of another one, i.e. how much knowledge
-



it has of the other component. [2] Coupling is usually contrasted with cohesion. Low coupling often correlates with high cohesion, and vice versa. Low coupling is often a sign of a well-structured computer system and a good design, and when combined with high cohesion, supports the general goals of high readability and maintainability. [3]

2) Keeping modules from circular dependency. A circular dependency is a relation between two or more modules which either directly or indirectly depend on each other to function properly. Such modules are also known as mutually recursive. In software design circular dependencies between larger software modules are considered an anti-pattern because of their negative effects - e.g. tight coupling, infinite recursions, unexpected compiling/linking failures and memory leaks. [4]

## 5.2 Module Implementation

### 5.2.1 UtilsRuntime

This library is to provide the implementation of smart pointer and higher level operation on thread. All the operations of a thread would seem as asynchronous tasks. In addition, scoped locking, ensuring that a lock is acquired when control enters a scope and the lock is released automatically when control leaves the scope [5], is provided to ensure the synchrony for accessing the same resource from different threads.

#### 5.2.1.1 CFinalize

Class Name: CFinalize

Namespace: Utils::SafeCoding

Description: Register a finalization in current scope. When the object of CFinalize destructed, it would execute the finalization code.

Usage:

```
int *pInt = new int;
CFinalize fin([]){ delete pInt; });
....
```

#### 5.2.1.2 IDisposable

Class Name: IDisposable

Namespace: Utils::SafeCoding

Description: A absolute virtual class providing a method Dispose to release the object.

---

### 5.2.1.3 CSmtPtr

Class Name: CSmtPtr

Namespace: Utils::SafeCoding

Description: A template class as a smart pointer. Using reference counter, it can easily find out which object should be released safely.

Macro: SREF

Usage:

```
CSmtPtr<ClassName> ref;  
SREF(ClassName) ref1;
```

### 5.2.1.4 IThread

Class Name: IThread

Namespace: Utils::Thread

Description: To provide a task based thread object. Developer can message a thread just using a function. The function would be running in the thread as a task. Also, it would return an asynchronous task object to control the task.

Factory: `IThread* CreateThread(const IThread::THREAD_CALLBACK& onInit = nullptr,  
const IThread::THREAD_CALLBACK& onFinish = nullptr);`

Relative Function: `IThread* GetCurrentThread();`

### 5.2.1.5 IAsyncTask

Class Name: IAsyncTask

Namespace: Utils:: Thread

Description: To provide a control of a task that is returned from a function invoke in a thread.

### 5.2.1.6 ILockable

Class Name: ILockable

Namespace: Utils::Thread

Description: To provide a basic interface of a lock. All locks should derivate from this interface.

### 5.2.1.7 ICriticalSection

Class Name: ICriticalSection

---

Namespace: Utils::Thread

Description: A multiple exclusive lock.

Factory: `ICriticalSection* CreateCriticalSection();`

#### **5.2.1.8 CScopeLock**

Class Name: CScopeLock

Namespace: Utils::Thread

Description: To lock a lockable object safely in a scope and release the lock after

Macro: SLOCK

### **5.2.2 XMLParser**

This module is an element in Data Layer. XML is used for formatting the data of communication. Meanwhile, it is also used for configuration of the upper level application. Therefore, a XML Parser is indispensable in the system. This library is to coding/decoding XML content conveniently. It treat the items and elements of XML contents as objects. There are three different class to describe the XML objects.

#### **5.2.2.1 IXMLDoc**

Class Name: IXMLDoc

Namespace: ESBXMLParser

Description: An interface that manage the document of a XML object. It would provide methods to load and save a XML file or content, and help developer to create the root node of a XML object.

Factory: `IXMLDoc* CreateXMLDoc();`

#### **5.2.2.2 IXMLNode**

Class Name: IXMLNode

Namespace: ESBXMLParser

Description: This interface is to provide the method to create, modify, and read content, elements, and attributes of a node of a XML object.

Factory: `IXMLNode* CreateXMLNode();`

#### **5.2.2.3 IXMLNodeList**

Class Name: IXMLNodeList

---

Namespace: *ESBXMLParser*

Description: Since one XML node would have more than one child nodes, this interface is provided to control the list of children of a XML node.

### 5.2.3 *ESBCommonLib*

This module is an element in Data Layer. It is to store basic structure information and static objects of the system. Also, it provides basic conversion between different structures.

### 5.2.4 *ESBWebService*

This module composes Communication Layer. This library provides network connection between server and client. It is based on gSoap, which is widely used for Web Services. Thus, the basic protocol between server and client would be HTTP/HTTPS. Security of communication can be ensured by OpenSSL.

The library would provide two different interfaces, *IESBWebServiceServer* for server and *IESBWebServiceClient* for client. Client would use method *Invoke* of *IESBWebServiceClient* to communicate with server, then server would get the request and call upper level module.

#### 5.2.4.1 *IESBWebServiceServer*

Class Name: *IESBWebServiceServer*

Namespace: *ESBWebService*

Description: To create a server in a C/S structure application. Developer can use this interface to create a listening to a TCP port. Once a client connects and sends its request, a function call would be made to upper module by using a callback function. All the requests would be processed asynchronously.

Factory: *IESBWebServiceServer*\* *CreateESBWebServiceServer()*;

#### 5.2.4.2 *IESBWebServiceClient*

Class Name: *IESBWebServiceClient*

Namespace: *ESBWebService*

Description: To provide methods to connect a server and to send a request to it.

Factory: *IESBWebServiceClient*\* *CreateESBWebServiceClient()*;

### 5.2.5 *ESBDataSerializer*

---

This module is an element in Basic Layer. This is a library providing us structure serialization. It has two forms of overloaded function. One is *Data2String*, another is *String2Data*. The previous function would serialize a data into a string and the following function would convert the string back to the data. Using the property of C++ language, the type of data would be checked in compiling time.

#### 5.2.5.1 Data2String

Function Name: Data2String

Namespace: ESBDataserialzer

Declaration: `bool Data2String(OUT std::wstring& string, IN <TYPE>& data);`

Description: To serialize a data into a string.

#### 5.2.5.2 String2Data

Function Name: String2Data

Namespace: ESBDataserialzer

Declaration: `bool String2Data(OUT <TYPE>& data, IN const std::wstring& string);`

Description: To convert a string back to a data.

### 5.2.6 ESBSMidService

This library represents Service Middle Layer in the architecture. Meanwhile some parts of stub layer is constructed in the library. This layer provides APIs for server programmer to develop services that would have the benefits from the system. Therefore, the programmer would be free from the consideration of load balancing and later connection binding.

#### 5.2.6.1 IESBService

Class Name: IESBService

Namespace: ESBSMidService

Derivate from: ESBSWebService::IESBSWebServiceServer

Description: To provide server interface to developers so that they can develop the server end point program of a C/S structure system. It provides basic requirements of a server application, including open a listening port. Meanwhile it provides methods to register the server to a Hub. The basic protocol of the middleware is HTTP/HTTPS, which is based on TCP/IP.

Factory: `IESBService* CreateESBService();`

---

### 5.2.6.2 IESBServiceHubConnection

Class Name: IESBServiceHubConnection

Namespace: ESBMidService

Description: To manage the connection between the server and the hub.

### 5.2.7 ESBMidConn

This library represents Client Middle Layer in the architecture. Also some parts of Stub Layer are constructed in the library. This library provides APIs for server programmer to develop client applications that would have the benefits from the system. The step of later connect binding is hidden in the layer. Therefore, developers would be free from later connection binding and load balancing.

#### 5.2.7.1 IESBConnection

Class Name: IESBConnection

Namespace: ESBMidClient

Description: To provide client interface to developers that for client end point programming of a C/S structure system. It provides methods to connect a server and send messages to the server. The basic protocol of the middleware is HTTP/HTTPS, which is based on TCP/IP.

Factory: [IESBConnection](#)\* CreateESBConnection();

### 5.2.8 ESBHubServiceLib

This library represents Hub Layer in the architecture. Some parts of Stub Layer are constructed in the library. This library is the logic implementation of Hub Layer, to separate itself from UI design. Load balancing and later connection binding is implemented in this library. It would process the registrations from services and token request from client. Moreover it would manage the services to promote the average usage rates of all the services by their loading statues.

#### 5.2.7.1 IESBHubService

Class Name: IESBHubService

Namespace: ESBHubService

Description: The logic implementation of Hub.

Factory: [IESBHubService](#)\* CreateESBHubService();

---

### **5.2.9 HubService**

The UI implementation of Stub Layer. This project would be compiled as an executable file that can be run in Windows. Meanwhile a configuration module is implemented in the HubService to customize the listening port.

## **6. System Test**

### **6.1 Sample Development**

To test the functionalities and stability of the system, it is necessary to develop a sample with the middleware. This sample is a database query application with the benefit of centralized service management and load balancing by using the APIs of the system, although from the view of developer it seems based on C/S structure. Thus, it has two components: client and server. Server would connect to database system and sent a client a query result when a query is received from the client.

#### **6.1.1 Server**

This application is used to measure the correctness of functionalities between Service Middle Layer and Hub layer. Moreover, the correctness of communication between Client Middle Layer and Service Middle Layer would also be checked and tested. The output of the application would show us the connection with the Hub and the session information from the client.

Using the Service Middle Layer, the server can be easily developed without consideration of communication between the Hub. Therefore, the total number of code lines of server is only 401 without comments or blank, including 91 lines of configuration parser, 141 lines of database connector, 120 lines of main module, and 48 lines of header files.

To connect to a database, the module of database connection uses the ADO library. Microsoft ActiveX Data Objects (ADO) enable client applications to access and manipulate data from a variety of sources through an OLE DB provider. Its primary benefits are ease of use, high speed, low memory overhead, and a small disk footprint. [6] Since the data source is from OLE DB provider, user can easily use different database systems by changing the connection string.

The server can be configured by a XML file, named “DQServer.xml”. We can create different services by changing the Service Port, the Service URL and the Service GUID. Database connection string and communication security can also be defined in the XML file.

#### **6.1.2 Client**

---

This application is used to measure the correctness of functionalities between Client Middle Layer and Hub layer, and communication between Client Middle Layer and Service Middle Layer. The output of the application would show us the SQL query result from the server.

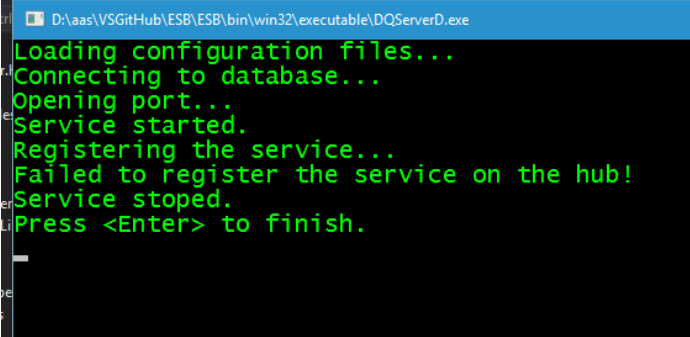
## 6.2 Sample Test

Following scenarios are used to test the correctness and stability of the system. We using an instance of the Sample Server to represent a Service of the system, and an instance of the Sample Client to represent a Client of the system. All the test results in scenarios should be the same or similar as the expected results, meanwhile any test cannot be passed with any crash(es) or runtime error(s).

### 6.2.1 Scenario 1. Run the Server while the Hub is not started/running.

**Expected Result:** The terminal of the server would print error message.

**Test Result:**

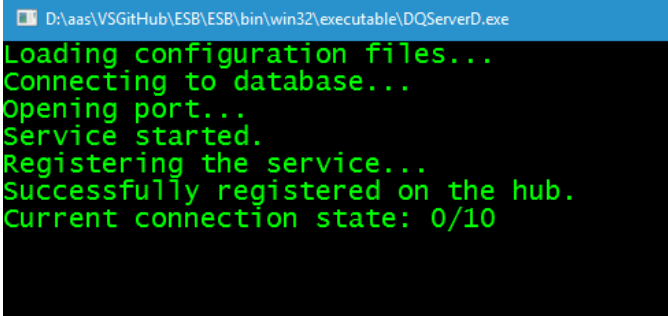


```
D:\aas\VS\Git\Hub\ESB\ESB\bin\win32\executable\DQServerD.exe
Loading configuration files...
Connecting to database...
Opening port...
Service started.
Registering the service...
Failed to register the service on the hub!
Service stopped.
Press <Enter> to finish.
```

### 6.2.2 Scenario 2. Run the Server when the Hub has already been started.

**Expected Result:** The terminal of the server would show successful registration message, and wait for the connections from clients.

**Test Result:**



```
D:\aas\VS\Git\Hub\ESB\ESB\bin\win32\executable\DQServerD.exe
Loading configuration files...
Connecting to database...
Opening port...
Service started.
Registering the service...
Successfully registered on the hub.
Current connection state: 0/10
```

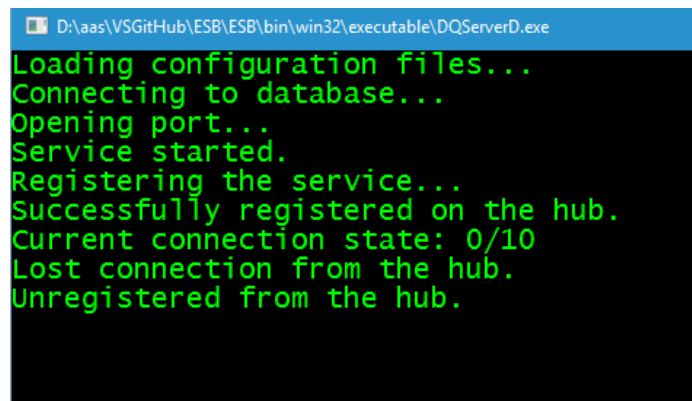
---



### 6.2.3 Scenario 3. Stop the Hub after the Server has already been registered on the Hub.

**Expected Result:** The terminal of the server would show connection lost message after a certain time.

**Test Result:**

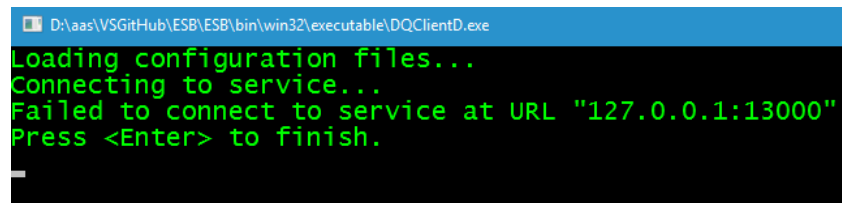


```
D:\aas\VS\GitHub\ESB\ESB\bin\win32\executable\DQServerD.exe
Loading configuration files...
Connecting to database...
Opening port...
Service started.
Registering the service...
Successfully registered on the hub.
Current connection state: 0/10
Lost connection from the hub.
Unregistered from the hub.
```

### 6.2.4 Scenario 4. Start a Client while the Hub has not been started.

**Expected Result:** The terminal of the client would show an error message.

**Test Result:**



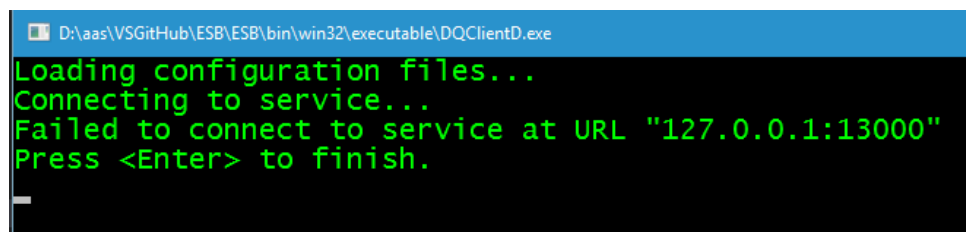
```
D:\aas\VS\GitHub\ESB\ESB\bin\win32\executable\DQClientD.exe
Loading configuration files...
Connecting to service...
Failed to connect to service at URL "127.0.0.1:13000"
Press <Enter> to finish.
```

### 6.2.5 Scenario 5. Start a Client while the Server has not been started.

**Prerequisite:** The Hub has already been started

**Expected Result:** The terminal of the client would show an error message.

**Test Result:**



```
D:\aas\VS\GitHub\ESB\ESB\bin\win32\executable\DQClientD.exe
Loading configuration files...
Connecting to service...
Failed to connect to service at URL "127.0.0.1:13000"
Press <Enter> to finish.
```

### 6.2.6 Scenario 6. Start a Client when the Hub and the Server have already been started.

---

**Expected Result:** The terminal of the client shows successful connection information and wait for user input of database query. The terminal of the service shows the confirmation of the client's session and the change of number of connections.

**Test Result:**

```

D:\aas\VS\GitHub\ESB\ESB\bin\win32\executable\DQServerD.exe
Loading configuration files...
Connecting to database...
Opening port...
Service started.
Registering the service...
Successfully registered on the hub.
Current connection state: 0/10
Session 16EBE9D0-A546-4E1F-A35C-F52C11C821C6 confirmed.
Current connection state: 1/10

D:\aas\VS\GitHub\ESB\ESB\bin\win32\executable\DQClientD.exe
Loading configuration files...
Connecting to service...
Connected successfully!

```

### 6.2.7 Scenario 7. Query database from the terminal of the client

**Prerequisite:** Scenario 6

**Expected Result:** The terminal of the client shows the result of query, including error messages that from database systems. The terminal of the Server shows the information of the queries from the client.

**Test Result:**

```

D:\aas\VS\GitHub\ESB\ESB\bin\win32\executable\DQServerD.exe
Loading configuration files...
Connecting to database...
Opening port...
Service started.
Registering the service...
Successfully registered on the hub.
Current connection state: 0/10
Session B721F5EA-597E-4250-A214-359294C8F096 confirmed.
Current connection state: 1/10
Session B721F5EA-597E-4250-A214-359294C8F096:select * from tb1
Session B721F5EA-597E-4250-A214-359294C8F096:select * from tb_33
Session B721F5EA-597E-4250-A214-359294C8F096:insert tb_33 values(4, 'efgn');
Session B721F5EA-597E-4250-A214-359294C8F096:select * from tb_33

D:\aas\VS\GitHub\ESB\ESB\bin\win32\executable\DQClientD.exe
Loading configuration files...
Connecting to service...
Connected successfully!
select * from tb1
=====
IDispatch error #3127
Invalid object name 'tb1'.
=====
select * from tb_33
=====
SQL Executed successfully!
   id  name
   --  ---
   1    abc
   2   abcde
   3  dddafds
3 results.
=====
insert tb_33 values(4, 'efgn');
=====
SQL Executed successfully!
1 item(s) effected.
=====
select * from tb_33
=====
SQL Executed successfully!
   id  name
   --  ---
   1    abc
   2   abcde
   3  dddafds
   4   efgn
4 results.
=====

```

### 6.2.8 Scenario 8. Query database from the terminals of two clients

**Prerequisite:** Scenario 6

**Expected Result:** The terminals of the clients show the result of query, including error messages that from database systems. The terminal of the Server shows the information of the queries from the clients.

**Test Result:**

```

D:\aas\VS\Git\Hub\ESB\ESB\bin\win32\executable\DQServerD.exe
Registering the service...
Successfully registered on the hub.
Current connection state: 0/10
Session A45D5D55-BB34-4188-ACF3-5B0750E76C1D confirmed.
Current connection state: 1/10
Session C7929534-349B-43E5-8310-4BEF7E2AD0F0 confirmed.
Current connection state: 2/10
Session C7929534-349B-43E5-8310-4BEF7E2AD0F0:select * from tb1
Session A45D5D55-BB34-4188-ACF3-5B0750E76C1D:select * from tb_33

D:\aas\VS\Git\Hub\ESB\ESB\bin\win32\executable\DQClientD.exe
Loading configuration files...
Connecting to service...
Connected successfully!
select * from tb_33
=====
SQL Executed successfully!
      id      name
      --      ---
      1         abc
      2        abcde
      3       dddafds
      4         efgn
4 results.
=====

D:\aas\VS\Git\Hub\ESB\ESB\bin\win32\executable\DQClientD.exe
Loading configuration files...
Connecting to service...
Connected successfully!
select * from tb1
=====
IDispatch error #3127
Invalid object name 'tb1'.
=====

```

### 6.2.9 Scenario 9. A Client connects to the Server whose connection pool is full.

**Expected Result:** The terminal of the Client shows error message. The Server would not confirm the new connection.

**Test Result:**

```

D:\aas\VS\Git\Hub\ESB\ESB\bin\win32\executable\DQServerD.exe
Session 4876198D-1FDD-4DC1-AA1B-FAD2492DED82 confirmed.
Current connection state: 1/10
Session 4876198D-1FDD-4DC1-AA1B-FAD2492DED82 end.
Current connection state: 0/10
Session DE38F6AF-8A38-47C4-8485-368D6A70864D confirmed.
Current connection state: 1/10
Session D11639E5-63EC-44D1-A19D-49E3DD7D2D04 confirmed.
Current connection state: 2/10
Session C4AD08A8-3492-4EF9-876F-205216BC291B confirmed.
Current connection state: 3/10
Session 262D2388-DBC3-47FA-A99E-FE671A56D1C3 confirmed.
Current connection state: 4/10
Session 8639E62C-21B8-4FF0-B19E-2047E94FCBCE confirmed.
Current connection state: 5/10
Session 672EECC-0BC2-480A-8C6B-0D04CFF04572 confirmed.
Current connection state: 6/10
Session 97E657A2-F8A1-4C57-98E1-8CD616D72870 confirmed.
Current connection state: 7/10
Session 26DFEB46-DD85-43BF-9D04-550BE214F778 confirmed.
Current connection state: 8/10
Session B9CACD7F-3087-4754-87CD-9F95F14E5107 confirmed.
Current connection state: 9/10
Session 18752AA6-0C2C-4789-9BF5-4E854AA7C1E5 confirmed.
Current connection state: 10/10

D:\aas\VS\Git\Hub\ESB\ESB\bin\win32\executable\DQClientD.exe
Loading configuration files...
Connecting to service...
Failed to connect to service at URL "127.0.0.1:13000"
Press <Enter> to finish.

```

### 6.2.10 Scenario 10. A Client connect to the Server whose connection pool is not full.

**Prerequisite:** After successfully test Scenario 9, one of the clients quit.

**Expected Result:** The terminal of the server would show successful registration message, and wait for the connections from clients.

**Test Result:**

```

D:\aas\VSGitHub\ESB\ESB\bin\win32\executable\DQServerD.exe
Session DE38F6AF-8A38-47C4-84B5-368D6A70864D confirmed.
Current connection state: 1/10
Session D11639E5-63EC-44D1-A19D-49E3DD7D2D04 confirmed.
Current connection state: 2/10
Session C4AD08A8-3492-4EF9-876F-205216BC291B confirmed.
Current connection state: 3/10
Session 262D2388-DBC3-47FA-A99E-FE671A56D1C3 confirmed.
Current connection state: 4/10
Session 8639E62C-21B8-4FF0-B19E-2047E94FCBCE confirmed.
Current connection state: 5/10
Session 672EEEC-0BC2-480A-8C6B-0D04CFF04572 confirmed.
Current connection state: 6/10
Session 97E657A2-F8A1-4C57-98E1-8CD616D72870 confirmed.
Current connection state: 7/10
Session 26DFEB46-DD85-43BF-9D04-550BE214F778 confirmed.
Current connection state: 8/10
Session B9CADC7F-3087-4754-87CD-9F95F14E5107 confirmed.
Current connection state: 9/10
Session 18752AA6-0C2C-4789-9BF5-4E854AA7C1E5 confirmed.
Current connection state: 10/10
Session 18752AA6-0C2C-4789-9BF5-4E854AA7C1E5 end.
Current connection state: 9/10
Session 2E2A3F2F-DA55-4EA0-9D68-18FED6FD93D5 confirmed.
Current connection state: 10/10

D:\aas\VSGitHub\ESB\ESB\bin\win32\executable\DQClientD.exe
Loading configuration files...
Connecting to service...
Connected successfully!

```

### 6.2.11 Scenario 11. Load Balancing

**Prerequisite:** Start 2-3 Servers with the same GUID while with the different URLs. Spontaneously execute a Client or quit a Client that already executed.

**Expected Result:** The terminals of the Servers would show the rates of usages are balanced.

**Test Result:**

```

D:\aas\VSGitHub\ESB\ESB\bin\win32\executable\1\sv1_1\DQServerD.exe
Loading configuration files...
Connecting to database...
Opening port...
Service started.
Registering the service...
Successfully registered on the hub.
Current connection state: 0/10
Session 6588797C-46AA-44B9-9779-72764831F9CC confirmed.
Current connection state: 1/10
Session 567BAADC-C885-4086-9F95-269B7C52637D confirmed.
Current connection state: 2/10
Session 95A14D94-CDA0-4696-A4FB-66920A2B1490 confirmed.
Current connection state: 3/10
Session F00EB637-5A10-4C42-9B22-D473C8443773 confirmed.
Current connection state: 4/10

D:\aas\VSGitHub\ESB\ESB\bin\win32\executable\1\sv1_1\DQServerD.exe
Loading configuration files...
Connecting to database...
Opening port...
Service started.
Registering the service...
Successfully registered on the hub.
Current connection state: 0/10
Session 9961B94D-A59A-428F-BE80-3DD18694BC93 confirmed.
Current connection state: 1/10
Session A5C9490B-E78D-41BC-B4B3-FDB809D6AE86 confirmed.
Current connection state: 2/10
Session AD76ED16-5024-4AA7-9503-CDD1136E3928 confirmed.
Current connection state: 3/10
Session AD76ED16-5024-4AA7-9503-CDD1136E3928 end.
Current connection state: 2/10
Session B268B8E7-3DEA-4EAS-A7AE-2716841F59CF confirmed.
Current connection state: 3/10
Session C0107DB9-AB34-4C2C-AE91-659CA6ECFFBE confirmed.
Current connection state: 4/10

```



The numbers of lines of code are generated by CLOC [7].

Language	files	blank	comment	code
C++	57	1616	989	24294

C/C++ Header	94	1189	619	6126
Total	151	2805	1608	30420

## 8. Future Works

### 8.1 Optimizing the Implementation of Load Balance

Current implementation of load balance is to reorder a vector in each time the Service information updates. However, it might be inefficient when there would be a huge number of Services in same type registered on the same Hub, for each updating request of service information would cost at least  $O(n \log n)$  CPU time, where  $n$  represents the number of the services. Therefore, to improve the efficiency of load status updating, the implementation of load balance should be optimized or redesigned.

### 8.2 Using Lock-Free Hash Map Instead of `std::map`

Standard C++ library provides `std::map` for mapping. However, it has the following two issues:

- 1) The time cost of query on `std::map` is  $O(n \log n)$ . It is much slower than hash map, which runs in  $O(n)$  time.
- 2) `std::map` is not safe for multi-threaded programming. Therefore, a lock is required to access a `std::map` object safely in several threads. However, once a thread owns the lock, other threads would be blocked if they also need the lock. The efficiency of multi-thread would be decreased drastically.

Lock-free hash map is suggested to solve the 2 issues of `std::map`. However, since there's no standard implementation of lock-free hash map, future study of the different implementations is required.

### 8.3 Environmentally Independent XML Parser

The functions of current version XML Parser is depending on the Microsoft Core XML Services (MSXML) 3.0. For this dependence, the code would be impossible to be compiled without the environment of Windows, and the target platform would be limited on Windows.

### 8.4 Cross Platforms

Current version of the system can only be running in Windows. Other systems, e.g. Linux, Unix, and Mac OS should be considered in the future work.

### 8.5 APIs for Cross Languages

---

The system only provides APIs for development in C++ language. Other languages, e.g. C, Java, C#, VB, etc. should be considered in the future work.

## 10. References

- [1] "Base64 - Wikipedia, the free encyclopedia," Wikipedia, 11 March 2016. [Online]. Available: <https://en.wikipedia.org/wiki/Base64>. [Accessed 12 March 2016].
  - [2] BOJAN, "HIGH COHESION, LOOSE COUPLING - THE BOJAN'S BLOG," [Online]. Available: <http://thebojan.ninja/2015/04/08/high-cohesion-loose-coupling/>. [Accessed 27 February 2016].
  - [3] "Coupling (computer programming)," Wikipedia, 9 November 2015. [Online]. Available: [https://en.wikipedia.org/wiki/Coupling\\_\(computer\\_programming\)](https://en.wikipedia.org/wiki/Coupling_(computer_programming)). [Accessed 27 February 2016].
  - [4] "Circular dependency," Wikipedia, 18 September 2015. [Online]. Available: [https://en.wikipedia.org/wiki/Circular\\_dependency](https://en.wikipedia.org/wiki/Circular_dependency). [Accessed 27 February 2016].
  - [5] D. C. Schmidt, "Strategized Locking, Thread-safe Interface, and Scoped Locking: Patterns and Idioms for Simplifying Multi-threaded C++ Components (1999)," 1999.
  - [6] "Microsoft ActiveX Data Objects (ADO)," Microsoft, [Online]. Available: [https://msdn.microsoft.com/en-us/library/ms675532\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/ms675532(v=vs.85).aspx). [Accessed 28 February 2016].
  - [7] A. Danial, "CLOC -- Count Lines of Code," [Online]. Available: <http://cloc.sourceforge.net>. [Accessed 28 February 2016].
  - [8] "What is Application Program Interface (API)? Webopedia," QuinStreet Inc., [Online]. Available: <http://www.webopedia.com/TERM/A/API.html>. [Accessed 29 February 2016].
-