Journal of Information Technology and Computer Science
Volume 2, Number 1, 2017, pp. 41-65
Journal Homepage: www.jitecs.ub.ac.id

Framework Design for Modular Web-based Application Using Model-CollectionService-Controller-Presenter (MCCP) Pattern

Aryo Pinandito[1], Ferdika Bagus Pristiawan Permana[2], Rizal Setya Perdana[3]

[1] Information System, Computer Science Faculty,
[2] Information and Communication Technology Division,
[3] Informatics Engineering, Computer Science Faculty,
Universitas Brawijaya, Malang, Jawa Timur, Indonesia

{aryo, ferdika.bagus, rizalespe}@ub.ac.id

Received 31 January 2017; accepted 22 June 2017

Abstract. Design pattern become an important thing in assisting the development of web-based application and in handling the problem of dynamically changing of organization structure and problem domain. Model-View-Controller (MVC) is a well-known design pattern in web-based application development due to its nature in the separation of an application into several parts, hence easy to reuse and maintain. However, integration of such design pattern requires several improvements in the design phase of information system development since it handles business process choreography and integration between application. Modification the interaction style of objects in a particular design pattern become a challenging problem. An application framework that employ Model-CollectionService-Controller-Presenter (MCCP) design pattern, which is resulting from modification of MVC design pattern, was proposed. The proposed framework also allows different applications to run simultaneously and provides inter-application data exchange mechanism to improve the data communication process between applications. The MCCP design pattern application framework shows that the application framework requires 51% less memory resources than another similar MVC framework such as CodeIgniter and cope the flexibility problem in data format presentations by providing a Presenter in the designed framework.

1. Introduction

Several organizations, which have many divisions or units, use several information system or computer application programs. The computer programs and information system were generally used to support their people daily work and organization business process. Many organizations develop custom applications in their information system in supporting their business processes by having their own in-house development team directly working and develop applications on-site. Several applications or computer programs were running in their own operating environment to meet organization or specific requirements of business processes.

In order to accommodate the changing dynamics of user requirements in a currently running business processes, web-based applications are more appropriate to be implemented than native desktop applications as users are not required to be manually install or update the application manually. In a web-based application, any changes made on the application server will be directly reflected and distributed to all active-users. As long as application users are using web-application using compatible web browsers there will be no issues in user's computer platform. Application developers will have full control in the distribution of developed application even though web-based application has limitation in programming code optimization, bigger overhead, and less control in utilizing computation resources of user computer than native desktop applications [1].

It is very common that applications need to be able to communicate with each other. There are cases that one application requires data from the output of the other application. The need of data integrations and business process choreography between units in organization urge the flexibility, ease of use, ease of access, and ease of development of information system in the future. The difference between applications data format, storage, and running environment make data integration between applications becomes difficult.

Although web-based applications in an information system are offering several more conveniences over native desktop applications, there are several other problems faced in a software development life cycle mainly in data integration problem between applications, user authentication, user authorization towards functionalities and data access among different applications. People who have several roles in organization business process may have different authorization towards different applications. That kind of users are often had to login and logout between applications in order to complete their tasks that needs to be performed in different applications.

Organization data stored in database are essentially owned and managed by their own respective application. Application has full control in managing their own data stored in a database. Such application is essentially not allowed to directly access nor modify data stored in a database owned by another application. Direct access or direct modification of other application data may cause high level coupling. Shared data access between application is categorized as common coupling or content coupling that is considered as a poor system architecture design [2]. High coupling level such as common and content coupling would lead to a decrease in terms of software quality. Therefore, a web-application framework design, which proposed in this research, had to accommodate data communication interface or Application Programming Interface (API) between applications that allow applications to exchange data among each other.

Another problem resolved in this research is related with user authentication and access control. When several users own several roles across applications, they may perform certain functionalities in several applications associated with their roles. Every role has their own respective functionalities among applications. Therefore, the designed application framework should allow users completing their task and perform several application functionalities in different applications based on associated roles without them having to re-authenticate while using different applications.

One of popular web-application design pattern is Model-View-Controller (MVC). MVC become very popular since several web application development frameworks were built on this pattern [3A]. In MVC design pattern, an application is divided into three parts, i.e. application logic as Controller, user interface as View, and application domain as Model. In MVC, reuse of programming code led to reduction in application code complexity, increasing code flexibility, and reduce coupling in application code components [4A]. However, MVC design pattern does not solve problem to be solved by programming code. MVC does help to write application programming code that is flexible, decoupled, easy to understand, and easy to reuse in another part of code [4A].

This research proposes an enhanced web application development framework based on MVC design pattern that allow application developers to develop multiple web-based applications in a single application framework setup in order to simplify the development process of applications and process of integrating the applications. The application framework also provides data communication between applications that allow application to access other application data. Several performance measurements were conducted to the framework in order to describe its performance characteristics compared to another similar web application framework.

2. Related Work

There are many choices of use framework in web development. Among of them, there is Laravel which is recently being very popular web application framework among web developers, Zend Framework which is devoted to enterprise scale web applications, Struts, JSF, Ruby on Rails, Grails, CakePHP, Django, Lift Framework, Catalyst, and many more. Every framework is offering their own advantages and disadvantages in particular aspects. Several issues faced by web application frameworks were memory management issues, complex design pattern, execution performance, scalability, maintainability, reusability, suitability, fitness for a particular purpose, and application modularity problem. Thus, leading software engineers and researchers to creatively propose solutions to resolve the issues.

Dragos-Paul Pop [5A] proposed a solid framework that is claimed to be able to reduce web application development times drastically, thus allowing web application developers to focus on application specific tasks rather than wasting more time in trying to implement or extend well-known patterns and practices. However, the performance of rendering engine and support in several NoSQL systems were require improvements.

Several existing frameworks have complex design patterns whereas the complexity of framework and application design were not always comparable to its performance. Some files were included during script execution. To overcome this problem, Umi Sa'adah [6] uses a simple design pattern, namely Singleton and namespace or package in Java. This research yields a lightweight PHP framework, with Singleton design pattern, namespace, AJAX, and multiple databases.

In another research, Hossein Shams [7A] introduces MVCC (Model-View-Controller-Context) as an architectural pattern solution for software frameworks to overcome problems that were faced by application programmers in the application layer. The architectural pattern can be implemented in various frameworks thus yield an easy and rapid application development, reusable code, and development flexibility for the developers.

Application development best practices are activities, technical or important issues, which are identified by users in a specific context, that render excellent services and expected to achieve similar results in similar situations. Every framework has its own best practices whose aim is to facilitate the development of web applications. However, there are no current comparative analysis that identifies best practices in web application frameworks. María del Pilar Salas-Zárate [8A] identifies a set of best practices for web frameworks. Afterwards, these best practices were analyzed and discussed in terms of developing Lift-based web applications. The identification of these best practices would allow developers to construct more interactive and efficient Lift-based web applications, integrating features found in Web 2.0 technologies with less effort thus exploiting framework's benefits.

3. Literature Review

3.1 Model-View-Controller (MVC)

Web-based Model-View-Controller (MVC) is a well-known design pattern used to develop a web based application that separates application into three parts based on business logic (Model), rendering output (View), and glue between Model and View (Controller). Reusing a Model by several Views is one of major advantages of MVC design pattern. Therefore, it is possible that an equal data to have multiple different presentations. MVC design pattern encourages developers to partition their application code into modular Model, View, and Controller. Therefore, application codes are relatively easy to reuse, maintain, and developed further.

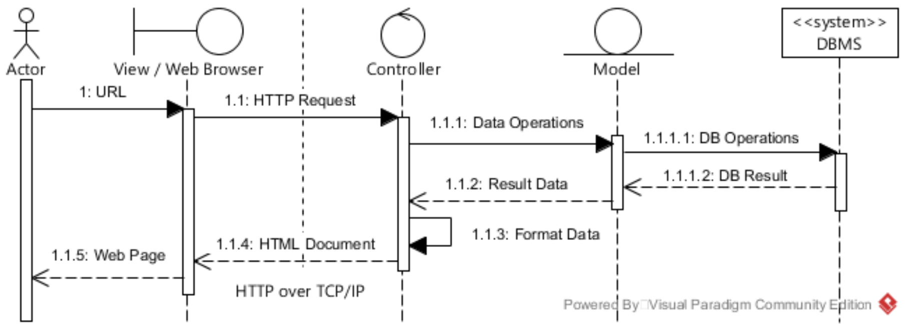

Fig. 1. Sequence diagram of MVC design pattern in web-based application.

View represents the functionality of presentation layer in an MVC architecture. It is basically containing user interface (UI) components were data elements were taken into a particular presentation. Model is responsible in managing all business logic and data structure that is required by business organization. It also handles all data requests from Controller through Views. A Controller, which maps application Model and View, describes the application logic process. Web-based application, which developed based on specific design pattern like MVC offers several advantages, such as application components reuse and provide good design consistency [9A]. Developing a web-based application using MVC design pattern allows easy maintenance and the possibility to an independent development by several different programmers [10A].

More or less, MVC design pattern is a direct translation to Boundary-Control-Entity objects in object-oriented analysis and design. In MVC design pattern, attributes and behaviors of entities or objects were translated into Models. Entities or objects which handled in a particular operation might be a single object or a collection of objects. Behaviors related to a single object and a collection of object might be different and may cause process ambiguity. Several behaviors might be applicable for a single object but not for a collection of object and vice versa. Therefore, this pattern could be improved by separating models that process singular object and a collection of object.

3.2 Model-CollectionService-Controller-Presenter (MCCP)

According to sequence diagram of MVC design pattern as depicted in Figure 1, a Model is responsible to communicate with Database Management System (DBMS) server and fetches data result from database that is required by business processes logic or requested by users. The implementation of MVC design pattern in a web application development still faces a problem related to the Model functionality. Analysis phase in software development yields class diagrams from the problem domain analysis which will be translated into Model objects. Single or set of database result object, which implemented in same object of Model, may difficult to maintain when there is a change business process domain model. If a single object and object collections were modelled in the same Model class, then attributes in a class will become conceptually ambiguous between a single Model object and its collection.

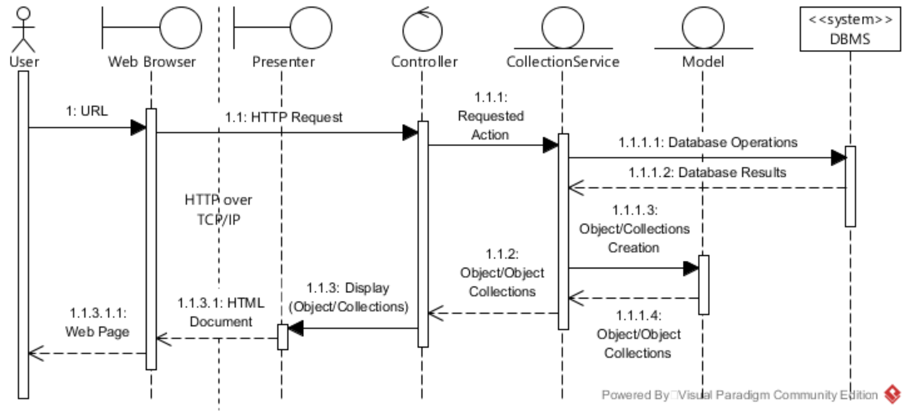
User
Web Browser
Presenter
Controller
CollectionService
Model
<<system>> DBMS
1: URL
1.1: HTTP Request
1.1.1: Requested Action
1.1.1.1: Database Operations
1.1.1.2: Database Results
1.1.1.3: Object/Collections Creation
1.1.1.4: Object/Object Collections
1.1.2: Object/Object Collections
1.1.3: Display (Object/Collections)
1.1.3.1: HTML Document
1.1.3.1.1: Web Page
HTTP over TCP/IP
Powered By Visual Paradigm Community Edition

Fig. 2. Sequence diagram of MCCP design pattern in web-based application.

Another problem related to the Controller-View in MVC design pattern is the implementation of a View presentation codes and Controller codes into a single instance Controller object. Instead of only organizing an application logic flows, a Controller also responsible to format the data obtained from Model into a particular document format. The consequences of this implementation, if a similar data structure from Model requires different presentation formats, a change in a Controller class is necessary to support the change.

A Model-CollectionService-Controller-Presenter (MCCP) design pattern allows web application developers to overcome these two main problems faced in MVC design pattern. The basic communication in MCCP design pattern is shown in Fig. 2. Data structures produced from class analysis will be implemented as Models, whereas CollectionServices implements Model's behaviors in database. Presenter object on MCCP design pattern is responsible in data presentation to clients, thus Controller objects were focused to handle the application and data logic flows of a client's request.

4. Methodology

The methodology used to develop the designed framework is following a standard software development life cycle (SDLC). It is including requirement analysis to obtain specific requirements related with common web-based application functionalities for a single HTTP request and the common pattern in providing services to users. The framework and its components as the solution were designed based on the analyzed requirements using object-oriented approach. The designed framework should be able to run several web applications as modules and allow them to communicate between each other to exchange data or information. In the implementation phase, the designed framework was implemented using PHP programming language. Several applications were also developed using the same programming language that run on top of the designed framework based on use case scenarios that were specified in previous phase. The implementation of framework and applications that were developed were tested against the specified functionalities and scenarios. Several performance measurements in terms of memory usage and processing speed were conducted and compared to another similar web application framework to depicts its characteristics.

4.1 Requirement Analysis

Analysis phase in software engineering yields software analysis models, e.g. software requirements specification, and logical analysis model in form of analysis classes and objects along with the relationship between classes or objects. Analysis models are produced by identifying objects in the problem domain along with their attributes, behaviors, as well as relationships among those.

Like most web-based application framework that uses the MVC design pattern, the designed web application framework should be able to identify the Controller to use, method of Controller to execute, and if any, given method parameters from the HTTP requests. In addition, the framework should be able to identify which web application to run from the requested URL as it designed to support multiple application to run over it. However, HTTP is a generic and stateless protocol [11A] that independently process HTTPS requests without any knowledge of previously executed request. Some web applications may track and retain user's state information from page to page by the use of HTTP cookies or server-side sessions. When multiple applications run for a single user session, it is possible that applications could see state information of user that belongs to another application for the same user session ID. By default, if an application knows the session ID of user, they will know session data associated it. Therefore, the framework should provide a mechanism to manage session data for a particular user and only allow application to create or modify their own session data.

In MCCP design pattern, identified classes and relationships between classes of objects in problem domain were modelled into Model classes. However, object behavior and data operations such as Create, Read, Update, and Delete (CRUD) were modelled into CollectionService classes. Therefore, there are no specific functionality requirements for base Model class except for required getter and setter methods of identified Model attributes. For an instance, there is a class abstraction for car object namely Car, any attributes related to car object and its relationship with other class were abstracted into a Model class of Car. Database CRUD operations associated with Car object or collection of Car objects were abstracted into class of CollectionService, say CarCollectionService class. However, one CollectionService class does not always related with one Model class. Depending on the circumstances, one CollectionService object may be associated with one or more Model class. As of CollectionService classes are highly related with database operations of Model, the proposed application framework should provide base class to provide basic database operations and certain functionalities. Therefore, application's CollectionService classes could simply inherit CollectionService class of framework to obtain certain database operation and functionalities.

In terms of data integration, every application might have different database settings or configurations as they have full authority over data in respective database of application even in determining the type of Database Management System (DBMS) used for their respective data operations and functionalities. If there is an application requires data from another application database, then it is necessary for the application, which own the required data, to provide a particular way or mechanism to verify the requests and provide the requested data. Based on the previously stated background analysis, the proposed framework provided in this research should be able to provide a uniform data functionality and allowing web-based applications to be designed and developed using a particular design pattern. In addition, the proposed framework should be able to provide a mechanism for applications to communicate between each other, manages user sessions, and manages inter-application sessions.

4.2 Framework Design

The framework was designed to support MCCP design pattern as previously depicted in Fig. 2. It separates each application functionality into Model, CollectionService, Controller, and Presenter parts as necessary. An architectural design model of the designed web application framework that is able to run multiple application is shown in Fig. 3. An application dispatcher behaves as an execution starting point of an application and responsible in identifying which application to run, controller to instantiate, method to execute, and parameters to be given from an HTTP request. The proposed framework also provides a database interface for common RDBMS used in web-based application.

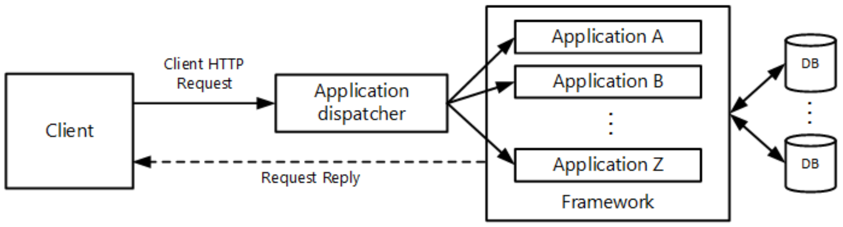

Fig. 3. Application framework high-level architectural diagram

A static view between web application components based on MCCP design pattern, the application framework, and relationships between them were modelled into class diagram as shown in Fig. 4. In the designed framework, application output or visual functionalities were handled by an instance of Presenter object inside a Controller.

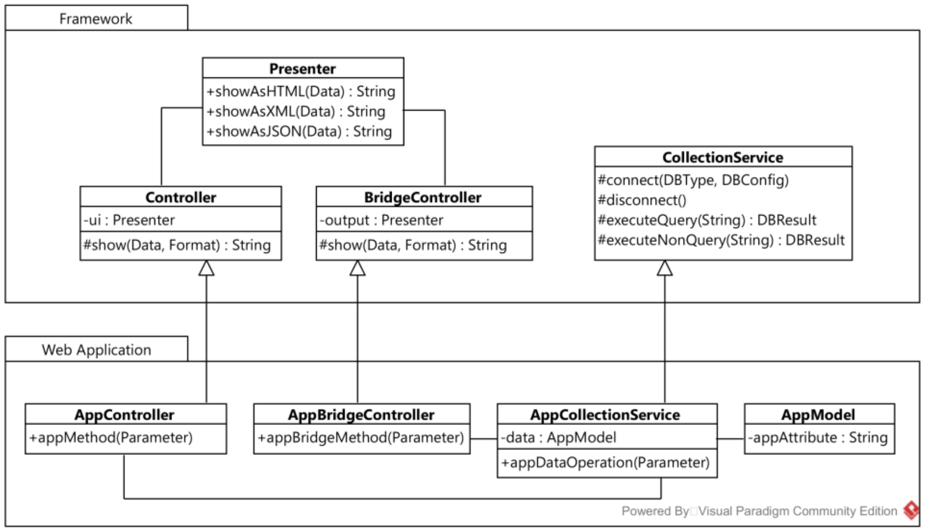
Framework
Presenter
+showAsHTML(Data) : String
+showAsXML(Data) : String
+showAsJSON(Data) : String
Controller
-ui : Presenter
#show(Data, Format) : String
BridgeController
-output : Presenter
#show(Data, Format) : String
CollectionService
#connect(DBType, DBConfig)
#disconnect()
#executeQuery(String) : DBResult
#executeNonQuery(String) : DBResult
Web Application
AppController
+appMethod(Parameter)
AppBridgeController
+appBridgeMethod(Parameter)
AppCollectionService
-data : AppModel
+appDataOperation(Parameter)
AppModel
-appAttribute : String
Powered By Visual Paradigm Community Edition

Fig. 4. Class diagram of MCCP application framework and class diagram of web application that runs on top of it.

Every application has their own unique application ID so that framework's application dispatcher is able to identify which application to run based on the application ID received request URL. The application ID should be composed of URL-compatible characters and it shouldn’t be too long for the sake of simplicity. Request URL format, which application dispatcher could parse for the designed framework, is shown in Fig. 5.

An application dispatcher main responsibility is to parse the received request URL and identify application to dispatch in terms of application components and data given that is separated by a forward slash from the request URL given. The mandatory components from the request URL is the application ID as the dispatcher requires it to determine which application to run. Basically, controller name and controller method to instantiate and execute are also compulsory. However, if the default name of application controller and method were already preconfigured, then if there are no controller name and/or method name were defined in the requested URL, the dispatcher will try to instantiate application's default controller, which has default controller name, and call its default method without giving any parameters.

In order to distinguish regular Controller and BridgeController call, application dispatcher detects in place of the controller name provided in the requested URL whether it is an "api" keyword. If so, then the application dispatcher will assume that the arguments right after the "api" keyword will be the bridge controller to instantiate. Hence, application controller cannot have name "api" as this name is used to distinguish between regular controller request and bridge controller request.

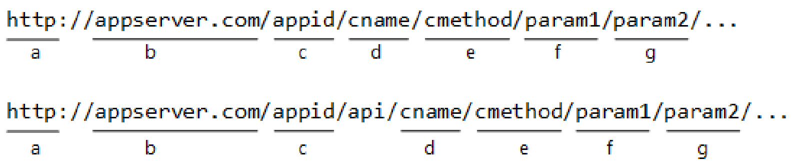
http://appserver.com/appid/cname/cmethod/param1/param2/...
a
b
c
d
e
f
g
http://appserver.com/appid/api/cname/cmethod/param1/param2/...
a
b
c
d
e
f
g

Fig. 5. Request URL format for a web browser client *(top)* and another application controller as client *(bottom)* that application dispatcher parse in determining application to run, controller to instantiate, controller method to call, and parameters to be given. Protocol scheme *a* used to identify the scheme used in the request; *b* is server host name; *c* is the application identifier; *d* is the controller name to instantiate; *e* is the method of controller to call; *f*, *g*, and so on are the optional method parameters.

Database Interface. The designed framework allows application to have their own DBMS type to use. However, in order to unify the way application to communicate with DBMS, CollectionService base class provide an interface to various DBMS Connector as depicted in Fig. 6. Concrete DBMS Connector class should realize the DBConnector interface and implements all generic database functionalities as defined in the interface. Hence, CollectionServices should not really concerned in writing a specific database code for a particular DBMS type. It is possible to provide further DBMS functionalities, i.e. transaction, by extending the DBConnector interface with another interface. However, not every DBMS support all functionalities that another DBMS have. Fig. 6. shows an application would be able application to perform database communications with three different DBMS types using a single CollectionService class without having to worry which DBMS type it connects to.

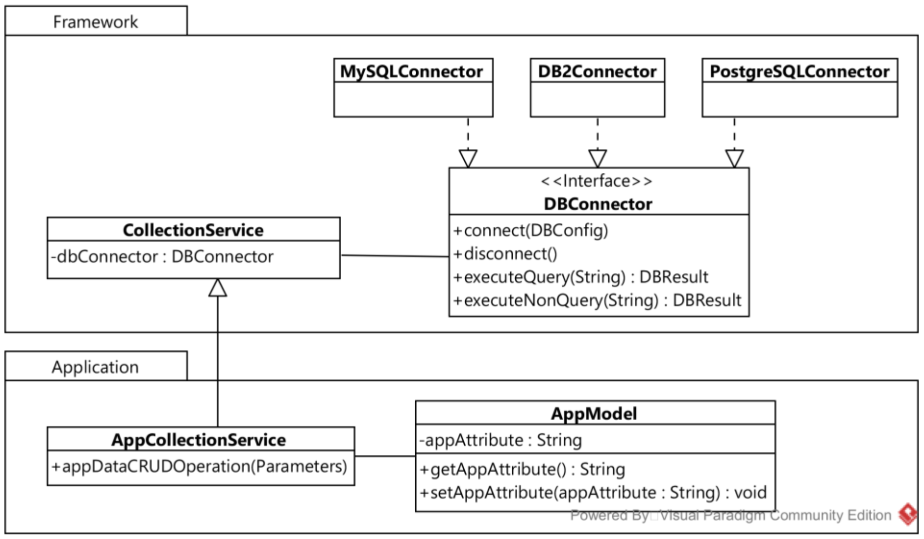
Framework
MySQLConnector
DB2Connector
PostgreSQLConnector
<<Interface>>
DBConnector
+connect(DBConfig)
+disconnect()
+executeQuery(String) : DBResult
+executeNonQuery(String) : DBResult
CollectionService
-dbConnector : DBConnector
Application
AppCollectionService
+appDataCRUDOperation(Parameters)
AppModel
-appAttribute : String
+getAppAttribute() : String
+setAppAttribute(appAttribute : String) : void
Powered By Visual Paradigm Community Edition

Fig. 6. Class diagram of application's CollectionService in providing generic database functionalities between different RDBMS.

The application database connectivity design as depicted in Fig. 6 shows the flexibility of an application to switch to another Relational DBMS type. However, there are drawbacks faced by the above design. If an application intends to use a non-relational or NoSQL database, i.e. NoSQL or MongoDB, then the database connectivity design would not work as expected. because there is no SQL query used in a NoSQL database engine. To resolve this issue, an application should implement their own database connectivity library to communicate with a NoSQL database.

Inter-application Communication. In order to communicate between applications, the designed framework provide a specific Controller to extend namely BridgeController that allow other application Controller to communicate with and exchange data. The BridgeController is designed to be separated with Controller as they both have different purpose even though they have similar behavior in the pattern. The separation is also intended to avoid confusion between Controller and BridgeController from the perspective of clients who perform the requests. The Controller is intended to serve HTTP requests from user's web browser while BridgeController is specifically intended to serve HTTP requests from Controller of another application. Data communication between applications diagram is shown in Fig. 7. Application BridgeController and Controller serve different type of client. They both implement different session handling method.

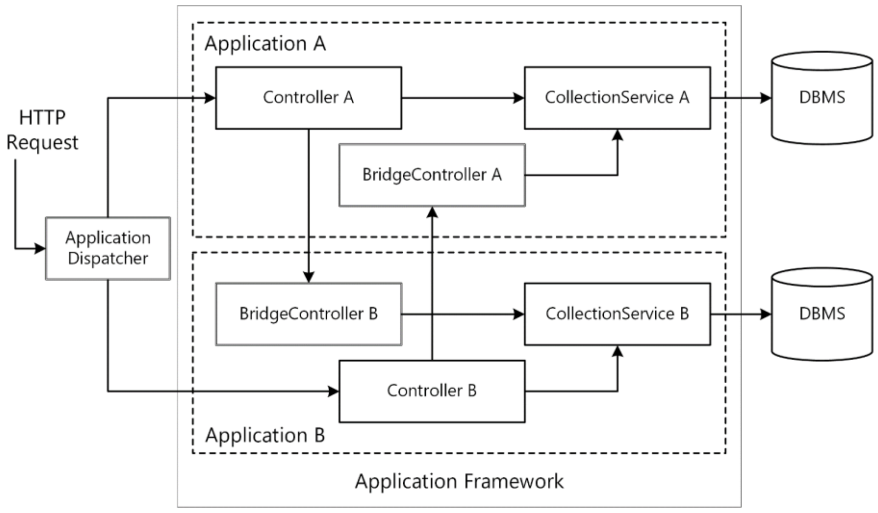

Fig. 7. Architecture diagram of inter-application data exchange communication.

Framework architecture diagram in Fig. 7 shows that the dispatched HTTP requests were served by a dispatched Controller. The Controller logically identify the required data sources to complete the request whether the data could be obtained from the database owned by the application or from the other application database. Application CollectionService responsible to provide data operations request from Controller or BridgeController. If a dispatched user request requires data from multiple data sources, then a Controller should be able to perform a data operation request to another application BridgeController. Communication between application Controller and another application BridgeController could be simply implemented using an HTTP Representational State Transfer (REST) architecture and Extensible Markup Language (XML) or JavaScript Object Notation (JSON) data format.

The sequence diagram for an application activity in processing a single HTTP request from a single user that requires data from multiple applications or data sources is shown in Fig. 8. As seen in the diagram, when an application Controller receives an HTTP request from a web browser client, it has two options in obtaining data. First, the Controller may obtain the required data from its own managed database using application's CollectionService. Second, the Controller could perform an HTTP request to another application that is served by another application's BridgeController. When an application CollectionService receives a data operation requests, it does not responsible to perform an execution authorization of a Controller. The authorization process of an HTTP request execution should be performed by application Controller or BridgeController before the actual CollectionService method is executed. Therefore, it is possible to reuse database-related methods in an application CollectionService by more than one Controller or BridgeController.

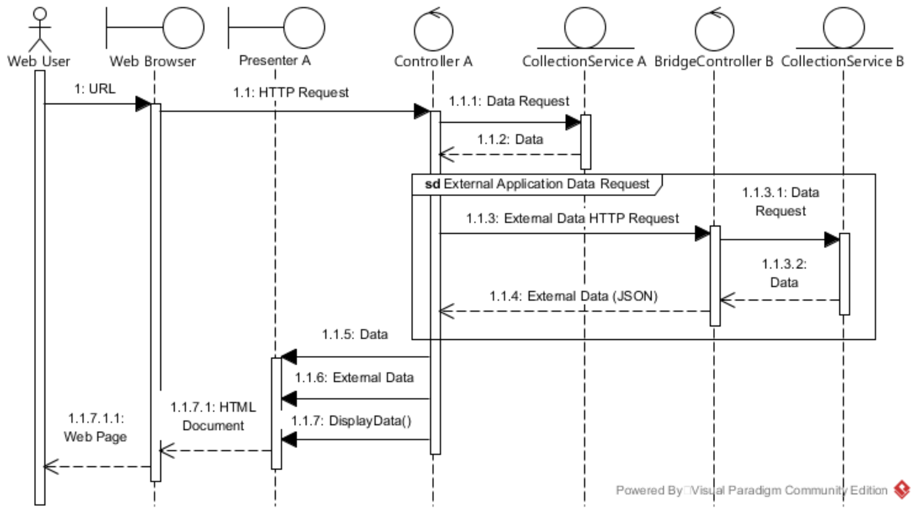

Fig. 8. Sequence diagram of multiple data source operation in a single HTTP request.

Communications between an application Controller and another application BridgeController are treated in a different way as an application Controller is not a web browser that simply accept cookies from application server to retain its states between requests. Furthermore, because inter-application communication was designed using standard HTTP protocol or REST, it is possible another type of clients, e.g. Unix-based console commands, native desktop applications, and mobile applications, are accessing the API through BridgeController. Therefore, it is important that an API provided by application BridgeController to have a particular mechanism in communicating with varying clients. The framework BridgeController also designed to communicate using a standard REST and returns a platform-independent plain-text document such as XML and JSON.

Session Handling. The framework is designed to handle requests to multiple applications simultaneously. If the session handling mechanism implemented in the usual single application way, therefore applications able to read or modify each other session data. Reading and modifying another application session data are certainly undesirable system behaviors. Therefore, the framework should provide a session handling mechanism that only allow application to read and modify their own session data.

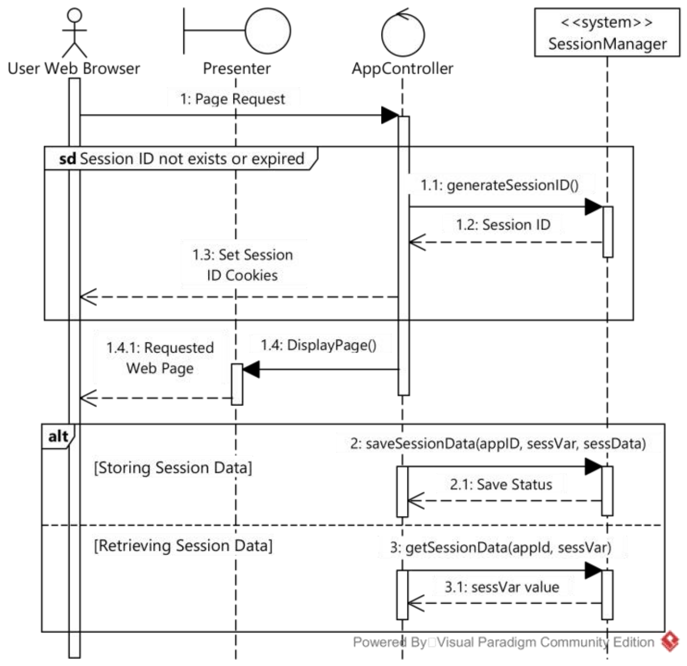
User Web Browser
Presenter
AppController
<<system>> SessionManager
1: Page Request
sd Session ID not exists or expired
1.1: generateSessionID()
1.2: Session ID
1.3: Set Session ID Cookies
1.4: DisplayPage()
1.4.1: Requested Web Page
alt
[Storing Session Data]
2: saveSessionData(appID, sessVar, sessData)
2.1: Save Status
[Retrieving Session Data]
3: getSessionData(appId, sessVar)
3.1: sessVar value
Powered By Visual Paradigm Community Edition

Fig. 9. Session handling management for a request from user web browser

The framework proposed a centralized session data storage to put every applications session data. The data may be stored in form of a DBMS storage or stored as files in a framework-accessible shared folder. Session data and their storage handling were managed by a SessionManager helper object. The SessionManager helper object will be accessible by application Controllers. In order to protect the session data stored from other applications, the SessionManager will mark any data stored on the session storage with application's ID of application that own the data. However, in order to protect the session data from application ID spoofing, therefore the SessionManager should obtain the application's ID from Application Dispatcher. The sequence on how the designed framework handles user session is shown in Fig. 10. The sequence diagram also shows involved objects of framework when they handle web page requests from web browser.

The design of session handling in inter-application (API) communication requires several steps. First, the requested application BridgeController should know which application is currently accessing the service. Each client application should be registered in a particular service API client database. Client applications should send their application ID in every request so that service API could identify its client. If a client identity of an API cannot be identified, then the framework returns an error message and the process should stop.

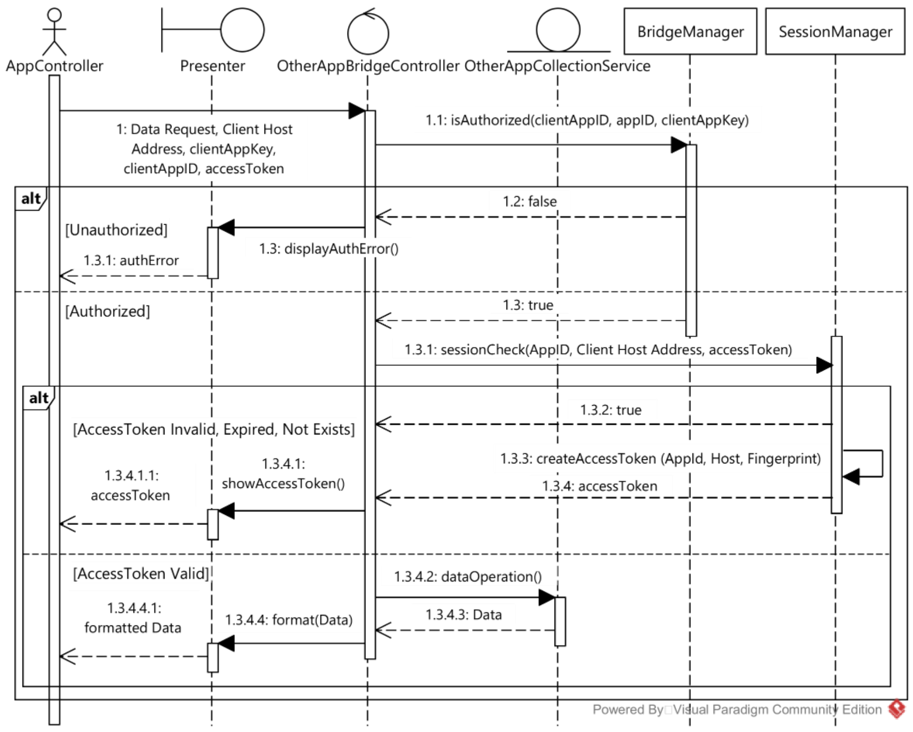
AppController
Presenter
OtherAppBridgeController
OtherAppCollectionService
BridgeManager
SessionManager
1: Data Request, Client Host Address, clientAppKey, clientAppID, accessToken
1.1: isAuthorized(clientAppID, appID, clientAppKey)
alt
1.2: false
[Unauthorized]
1.3: displayAuthError()
1.3.1: authError
[Authorized]
1.3: true
1.3.1: sessionCheck(AppID, Client Host Address, accessToken)
alt
1.3.2: true
[AccessToken Invalid, Expired, Not Exists]
1.3.3: createAccessToken (AppId, Host, Fingerprint)
1.3.4.1: showAccessToken()
1.3.4.1.1: accessToken
1.3.4: accessToken
[AccessToken Valid]
1.3.4.2: dataOperation()
1.3.4.4.1: formatted Data
1.3.4.4: format(Data)
1.3.4.3: Data
Powered By Visual Paradigm Community Edition

Fig. 10. Sequence diagram of external application data request.

In an API request, a client of an application API send a request URL along with its application key, application id, access token given, and data to send. If the client is unauthorized to access the service, then the framework will return authorization error messages, otherwise its access token will be checked against its validity. An access token is considered to be valid if its fingerprint is valid and its access token is within a valid timespan. A fingerprint is a hash value that is computed from client's hostname or IP address and client's application id. If client's access token given is invalid, i.e. not exists or expired, then as long as its computed fingerprint is still correct, then a new access token will be generated and returned to the requesting client. If both client's access token and its computed fingerprint is correct, then the application will continue to proceed client's requests. Authorization check sequence in an application BridgeController's will be processed in framework's BridgeController base class, thus application BridgeController just need to extends the class. Detailed sequence of involved objects inside the designed framework along with its responsibilities of an inter-application communication between multiple applications are shown in the sequence diagram in Fig. 9.

4.3 Implementation and Testing

The proposed framework designs were implemented in a virtualization environment of Linux Centos 7 operating system. A web-based environment was setup to run several web-based applications using an Apache web server with a PHP library as a module. Two RDBMS, i.e. PostgreSQL and MariaDB as a replacement for MySQL, were also setup and configured to represent a multi database server environment. The database server was setup in the same operating system environment as the web server, therefore configured data connectivity between web server and database server will be localhost.

The designed application framework, application dispatcher, and the application itself were implemented using a PHP programming language. Every request received by web server is converted into a particular URL format which directed to the application dispatcher to parse using Apache web server rewrite module. This way, the application dispatcher script will be hidden to the clients, thus creating more secure application framework.

In the following file and directory structure, the document root that is accessible to the clients is inside the /var/www/html directory. The framework base classes such as Controller, CollectionService, Presenter, and database connector classes were located in /var/www/framework directory. Therefore, it only accessible through application classes and codes. Application that run over the framework were located in the /var/www/html/apps directory and all of their application code were placed under their own application id and the code respective class directory, i.e. model classes in /model directory, controllers in /controller directory, collection services in /collectionservice directory, bridge controllers in /bridgecontroller directory, and application's UI files, which will be read by framework's Presenter class, located in /view directory.

Directory structure of the designed application framework that hosts two applications whose application ID appA and appB.

```
/var/www/framework/
/var/www/html/appdispatcher.php
/var/www/html/apps/
                   /appA/model/
                        /collectionservice/
                        /controller/
                        /bridgecontroller/
                   /view/
                   /appB/...
                   /...
```

Several implementation test scenarios used in this research were designed to follow the application testing structure as shown in Fig. 10. This research implements two applications, i.e. application A and B, that each application uses their own DBMS type. Application A uses MariaDB, while application B uses PostgreSQL as their data storage. Application A has one table of Student in MariaDB server. Application B also has one table of Book in PostgreSQL server. Both Student and Book have similar columns and numbers in their respective database server.

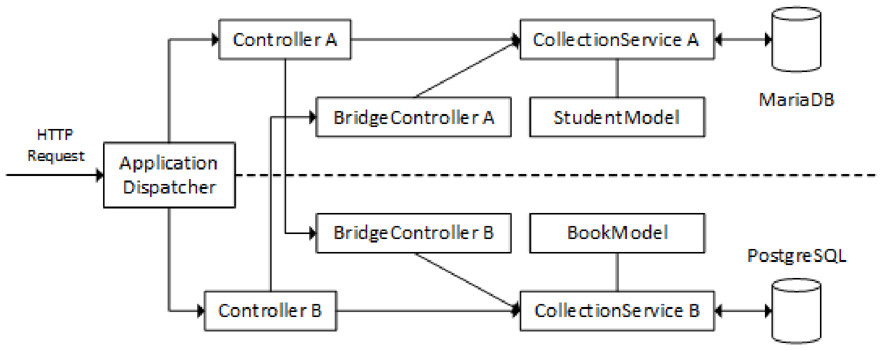

Fig. 11. Implementation and test environment setup

By using framework and application setup structure as depicted in Fig. 10, in order to depicts the application framework characteristics and evaluates its functionalities as previously designed, several test scenarios used in this research are:

1. Application A receive a request to display a list of students and put another student data that retrieved or written from/into a MariaDB database server. This scenario evaluates the Controller and CollectionService components of Application A functionalities in providing services to user request.
2. Application B receive a request to display a list of books and put another student data that retrieved or written from/into a PostgreSQL database. This scenario also evaluates the Controller and CollectionService components of Application B functionalities in providing services/data to user request from a different data sources.
3. Application B receive a request to display a list of students that retrieved from MariaDB database that is provided by Application A. This scenario evaluates both Controller of Application B and BridgeController of Application A communication which depicts the capability of one application to be able to interact with another application to obtain data/information that is owned or provided by another application or platform.

Test scenario number 1 and 2 from the above test scenarios are depicting the abilities of the designed framework to simultaneously run several applications and also showing that applications are able to perform database operations regardless the type of database used. Test scenario number 3 shows the designed framework abilities in providing data communication mechanism between different applications.

Below is an example of a Student object representation as Student Model class

Application A Student Model class named Student

```
<?php
class Student {
  public $id;
  public $name;
  public $email;
}
```

In the proposed design, a Model is a basic representation of problem domain object without concerns on how the Model object will communicate with database servers. Therefore, there is no need to implements data communication code with database server in a Model class as Model's database manipulation and operation functionalities will be independently implemented in application CollectionService classes. An implementation example of an application CollectionService is shown in below code:

Application A CollectionService class named StudentCollectionService

```
<?php
class StudentCollectionService extends
CollectionService {
    public function __construct() {
      parent::__construct();
    }
    public function readAll() {
      $sql = "SELECT * FROM students";
      $this->dbConnector = new
MySQLConnector($connectParam);
      $this->dbConnector->connect();
      $result = $this->dbConnector->query($sql);
      $students = array();
      foreach($result as $row) {

        $s = new Student();
        $s->id = $row['id'];
        $s->name = $row['name']
        $s->email = $row['email'];
        array_push($students, $s);
      }
      return $students;
    }
    public function save($id, $name, $email) {
      $sql = "INSERT INTO student (id, name, email) "
             ."VALUES ('$id', '$name', '$email') "
             ."ON DUPLICATE KEY update name = '$name', "
             ."email = '$email')";
      $this->dbConnector = new
MySQLConnector($connectParam);
      $this->dbConnector->connect();
      $result = $this->dbConnector->query($sql);
      return $result;
    }
  }
```

Application A Controller class named HomeController

```
<?php
  class HomeController extends Controller {
    public function __construct() {
      parent::__construct();
    }
    public function index() {
      $studentCs = new StudentCollectionService();
      $listOfStudents = $studentCs->readAll();
      $this->presenter->show('studentIndex.php',
        $listOfStudents);
    }
    public function save($id, $name, $email) {
      $studentCs = new StudentCollectionService();
      $status = $studentCs->save($id, $name, $email);
      if($status) $this->index();
      else $this->presenter->showError('Database
Error');
    }
  }
```

Examples of an implementation codes of an application BridgeController in PHP programming language, which allow other applications to retrieve all student data contained in MariaDB database of application, is shown below:

Application A BridgeController class named HomeBridgeController that allow other application to retrieve student data.

```
<?php
  class HomeBridgeController extends BridgeController {
    public function __construct() {
      parent::__construct();
    }
    public function getStudents() {
      $studentCs = new StudentCollectionService();
      $listOfStudents = $studentCs->readAll();
      $this->presenter->jsonEncode('studentIndex.php',
        $listOfStudents');
    }
```

Application B Controller class named HomeController that retrieves student data from Application A

```
<?php
  class HomeController extends Controller {
    public function __construct() {
      parent::__construct();
    }
    public function index() {
      $encodedData =
        $this->callApi('appA','home/index/getStudents');
      $listOfStudents =
        $this->presenter->jsonDecode($encodedData);
      $this->presenter->show('studentIndex.php',
        $listOfStudents);
    }
```

Test Result. Based on the test conducted to the scenario number 1 and 2, it is known that the proposed framework was able to run multiple applications simultaneously in a single application framework setup. The dispatcher for the application framework is able to identify application to run from a request URL given from users. Both applications are able to obtain and return the requested data in a particular format from two different DBMS type using MCCP design pattern.

On the test scenario number 3, a request to application B is trying to obtain the same data as test scenario number 1. However, in order to obtain the requested data from application A, application B communicates with application A through an API call. Based on the test, a client request to application B that requires data from application A can be successfully completed.

In order to measure the application framework performance characteristics, several runtime execution tests were done based on the test scenario number 1 and 3. During runtime execution tests, the execution time of a request and its memory usage were measured. The measurements were conducted for read-write requests of a single application and read-write requests that requires inter-application communication. The application framework execution time measurement results are shown in Table 1. The measurement tests were performed using Apache JMeter as application's client.

The proposed framework was deployed in a Linux Centos 7 as an Oracle VirtualBox guest operating system in Windows 10 Professional host. The guest operating system is configured to run with 1 core of CPU, with 1024 MB of memory. The host machine is running on a physical Intel Core i7-4790 CPU with 1TB of HDD, 8 GB of DDR3 RAM, and virtualization settings enabled in BIOS. Both application framework and database servers are installed and running inside guest operating system. Application framework software testing environment used during testing are Apache Web Server 2.2, PHP 5.6 library, MariaDB 5.5.50, and PostgreSQL 9.6.1.

Table 1. Application framework execution time performance measurement test results.

No	MCCP		MCCP Bridge	
	Read (ms)	Write (ms)	Read (ms)	Write (ms)
1	49.49	49.59	72.31	81.32
2	49.79	48.55	70.08	83.59
3	39.92	54.13	72.40	83.90
4	45.25	45.11	67.44	84.53
5	40.37	43.59	80.05	87.03
6	39.65	40.57	76.44	81.19
7	53.20	41.32	85.49	78.76
8	39.96	40.27	79.14	69.85
9	40.38	41.63	80.56	70.41
10	40.79	50.82	74.79	69.79
11	49.19	50.58	66.61	72.75
12	54.45	40.32	67.96	69.10
13	42.31	40.12	67.60	75.82
14	41.08	40.97	70.84	69.34
15	41.57	40.95	72.61	71.55
16	40.77	41.39	71.81	71.87
17	41.76	40.68	71.37	71.14
18	46.10	45.73	70.87	72.71
19	40.38	40.18	80.23	70.69
20	41.05	40.05	69.18	69.50
21	40.87	48.03	76.73	75.11
22	42.91	41.04	70.76	70.02
23	41.76	39.97	71.07	72.24
24	38.88	41.67	71.00	69.11
25	45.62	42.44	68.94	67.82
Average	43.50	43.59	73.05	74.37

Table 2. Application framework average memory usage.

	Read (kB)	Write (kB)
MCCP	1164.696	1135.240
MCCP Bridge	921.166	907.265

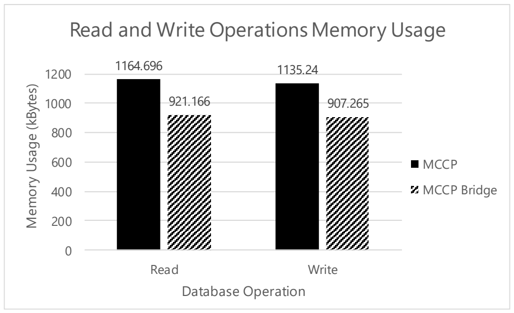

Fig. 12. Memory usage in read and write data operation for a single MCCP and MCCP Bridge communication scenarios.

Table 3. Read and write execution time comparison between the proposed application framework and CodeIgniter framework in milliseconds.

No	MCCP Framework		CodeIgniter	
	Read (ms)	Write (ms)	Read (ms)	Write (ms)
1	49.49	49.59	48.89	45.18
2	49.79	48.55	47.18	51.57
3	39.92	54.13	42.37	43.76
4	45.25	45.11	44.55	46.57
5	40.37	43.59	43.72	42.99
6	39.65	40.57	42.10	41.05
7	53.20	41.32	42.04	46.45
8	39.96	40.27	46.40	49.94
9	40.38	41.63	41.52	39.90
10	40.79	50.82	41.48	40.31
11	49.19	50.58	41.11	41.66
12	54.45	40.32	43.61	40.57
13	42.31	40.12	41.74	44.94
14	41.08	40.97	52.03	40.09
15	41.57	40.95	43.76	41.75
16	40.77	41.39	46.14	42.46
17	41.76	40.68	44.33	42.36
18	46.10	45.73	42.58	41.87
19	40.38	40.18	45.16	40.61
20	41.05	40.05	51.68	41.23
21	40.87	48.03	50.81	41.39
22	42.91	41.04	43.63	40.69
23	41.76	39.97	46.39	40.37
24	38.88	41.67	44.35	41.44
25	45.62	42.44	42.79	44.12
Average	43.50	43.59	44.82	42.93

Based on the execution time performance tests and comparison shown in Table 1, there is an additional 30 milliseconds in average of additional time overhead in an application request that requires an API call than a request that does not requires external application data. However, there are no significant execution time differences between read and write data operations for both scenarios. The memory usage between test scenario number 1 and 3 of the requested application is shown in Table 2 and Fig. 12. It shows that the memory usage in a single MCCP application request in a read or write data operation, requires around 1.1 MB of memory. A single MCCP application request that requires external application data uses memory for about 900 kB.

This research also conducts execution time and memory usage comparison between the proposed framework and the popular CodeIgniter framework. The time execution and the memory usage comparison between the proposed framework and CodeIgniter framework is shown in Table 3. Based on the test results shown in Table 3, both the proposed framework and CodeIgniter framework takes around 43ms in average to perform read or write data operation in a single application request. However, there are no significant execution time differences between the proposed MCCP application framework and CodeIgniter framework.

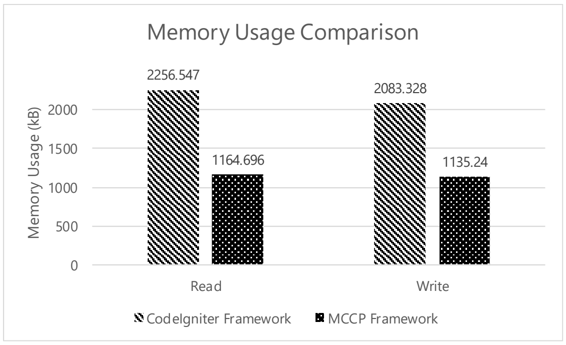

Fig. 13. Memory usage comparison in read and write data operation between MCCP and CodeIgniter framework in kilobytes.

Based on the application memory usage comparison between the proposed MCCP application framework and CodeIgniter as shown in Table 1, there are significant memory usage differences for a single read and write data operation request. The proposed framework uses 1.1 MB of memory while the CodeIgniter framework requires more than 2000 kB of memory in both read and write data operation scenarios for a single request. The software version of CodeIgniter framework used in the comparison test is v3.1.3.

4. Result and Discussion

In this paper, we have demonstrated the implementation of a new approach web-based design pattern MCCP. The proposed design pattern is able to communication between application and more flexible due to the change of the business process requirements. Such representations are advantageous for many reasons. For example, the proposed design pattern in this research is designed based on real world organization problems in a highly changing requirement environment and developed on the basis of the needs of data integration between different applications within the organization.

The test results in this MCCP design pattern application framework shows that the application framework use less resources than another similar MVC framework such as CodeIgniter in terms of memory usage. Furthermore, MCCP design pattern also cope the flexibility problem in data format presentations by providing a Presenter in the designed framework. Based on the test conducted, the designed framework is capable to run multiple applications simultaneously in a single framework setup.

In overall, MCCP-based modular application framework is successfully implemented and tested with some drawbacks. First, this design pattern requires more performance, scalability, and several other non-functional evaluations before being used in a large-scale production environment in order to know the behavior of the application framework in a large-scale environment. Second, despite the interrelation is assumed in same organization, it has not implements an authentication mechanism between application. Third, the possibilities in scaling the application server physical setup yields session handling and application handling problems. Fourth, when several applications run by a single application framework and problems occurred in the application framework itself instead of the application that run on top of it, then every application that is currently running will be unstable.

Based on the measurement test results, a single application request that requires external application data requires less memory. This is due to an application Controller that requires data from external applications, it performs an external API call to obtain the requested data instead of instantiating a CollectionService. However, despite of using less memory, the external applications that provide the necessary data from the API call in the second scenario also uses memory as in the first scenario. Therefore, if the memory usage of both applications that communicate were summed, then they both requires more memory than in a single application usage scenario. Security issue in inter-application communication should be taken into concern. As the proposed communication mechanism offer several possibilities to access an application API from different type of application as its clients.

Future enhancements can be performed by adding a particular User Access Control (UAC) mechanism based on defined application user roles such as Role-Based Access Control (RBAC) thus improving security in the mechanism of data access. Another enhancement to the designed framework is the ability in managing running applications during runtime as multiple applications will run on top of it. Instability of application will affect the overall stability of other applications. Therefore, a dedicated Operational and Maintenance (OAM) systems that allow system administrator to monitor the runtime condition and manage every application that run over the framework would be highly necessary.

5. Conclusion

This paper presents design of a web application framework that allow multiple application run on top of it. The framework implements Model-CollectionService-Controller-Presenter (MCCP) design pattern. It enables application to provide web services or API and exchanges data between applications through API call mechanism which handled by BridgeControllers of application. Application request between from a web browser and an application Controller as clients were handled differently.

There are no noticeable differences in terms of execution time between the proposed framework design and CodeIgniter application framework. The proposed application framework had a similar performance in terms of the execution time required to process a single request. However, the proposed application framework requires less memory to process a single data operation request from a client.

Inter-application communication between an application Controller and another application BridgeController has 30ms execution time overhead on a single application framework local setup. Despite of less memory usage of the requested application in this inter-application communication scenario, there is an additional memory requirement for the other requested application in processing other application the request. In other words, when an application calls another applications API, it is basically the same as running two or more applications at the same time, thus requires more computing resources. However, data exchanges between applications is made possible with the proposed mechanism in the designed framework.

In larger context, there are possibilities in implementing the proposed framework design into mobile platform so that a single mobile application could provide different and modular services as required by its users. However, the implementation of the designed framework in mobile platform could be analyzed and researched further to obtain several findings and depicting its characteristics. System scaling in terms of data capacity and performance-related problems could be an interest of further research, analysis, and improvement.

Acknowledgments. We would like to convey our biggest appreciation to all members of Application and Integration System division of Information and Communication Unit in Universitas Brawijaya and all our colleagues who contributes and share their thoughts into our works.

References

1. Youngwerth, A.: What are the advantages and disadvantages of web based application development vs. desktop application development?. Quora, https://www.quora.com/What-are-the-advantages-and-disadvantages-of-web-based-application-development-vs-desktop-application-development
2. Vliet, H. v.: Software Engineering: Principles and Practice. Wiley, (2007)
3. Arora, S.: PHP Frameworks: The Best 10 for Modern Web Development, " in NoeticForce. NoeticForce, http://noeticforce.com/best-php-frameworks-for-modern-web-development.
4. Meeus, J.: MVC – a Problem or a Solution?. Sitepoint, https://www.sitepoint.com/mvc-problem-solution/.
5. Pop, D.: Designing an MVC Model for Rapid Web Application Development. (2014)
6. Sa'adah, U.: Implementing Singleton method in Design of MVCBased PHP Framework. (2015)
7. Shams, H.: MVCC: An Architectural Pattern for Developing Context-aware Frameworks. (2014)
8. Salas-Zárate, M. d. P.: Analyzing best practices on Web development frameworks: The lift approach. (2015)

9. Sridaran, R, Padmavathi, G, Iyakutti, K: A Survey of Design Pattern Based Web Applications. Journal of Object Technology, vol. 8, no. 2. (2009) 61-70.

10. TM Staff: What is MVC Architecture in a Web-Based Application?. Trademark Productions [https://www.tmprod.com/blog/2012/what-is-mvc-architecture-in-a-web-based-application/

11. The Internet Society Hypertext Transfer Protocol -- HTTP/1.1, https://www.w3.org/Protocols/rfc2616/rfc2616.txt