Journal of Software Engineering

ISSN 1819-4311

Academic Journals Inc.

www.academicjournals.com

Journal of Software Engineering 8 (1): 1-13, 2014
ISSN 1819-4311 / DOI: 10.3923/jse.2014.1.13

A Configurable and Extensible Middleware Design for Mobile Application Integration

[1]Nien-Lin Hsueh, [1]Der-Hong Ting, [2]Wen-Hsiang Shen and [3]Wen-Tin Lee

[1]Department of Information Engineering and Computer Science, Feng Chia University 100 Wenhwa Road, Taichung, 40724, Taiwan, Republic of China

[2]Department of Information Management, Overseas Chinese University, 100, Chiaokwong Road, Taichung 40721, Taiwan, Republic of China

[3]Department of Software Engineering, National Kaohsiung Normal University, 62, Shenjhong Road, Yanchao District, Kaohsiung, 82446, Taiwan, Republic of China

Corresponding Author: Wen-Hsiang Shen, Department of Information Management, Overseas Chinese University, 100, Chiaokwong Road, Taichung 40721, Taiwan, Republic of China Tel: 886427016855 Fax: +886427075420

ABSTRACT

In the recent years, more and more corporations are devoted to developing mobile applications integrated with legacy systems in pace with rapid development of mobile devices. Mobile applications can attract potential customers to increase the competitiveness of corporations and promote working efficiency for employers. However, it is difficult to develop mobile applications integrated with legacy systems because of many heterogeneous systems and diverse mobile platforms. Moreover, the cost of developing and maintaining such systems is very high. Thus, applying mobile middleware can facilitate in resolving most of the problems. The main objective is to build a communication bridge between the legacy systems and diverse mobile platforms and to setup the common communication protocols and data formats for different mobile platforms. In our research, to facilitate the extension of modules used for integrating with the legacy systems, the Plugin design pattern has been applied to mobile middleware and further implemented many common modules for various functions. The developers are able to create the Plugin components needed for the existing mobile systems rapidly. Finally, we apply the framework to develop FCU mobile middleware in order to assist the university develop mobile services that can be integrated with the legacy systems and promote current mobile application development extensibility.

Key words: Middleware, legacy system evolution, design pattern, configurable design

INTRODUCTION

In the recent years, the smart mobile devices have been developed rapidly and according to the Internet data center's research report, in 2011 the global shipments of mobile devices reached 450 million. In addition, the number of mobile application download is over 24.9 billion which is estimated to reach 183 billion in 2015 (Wauters, 2012). Besides, mobile applications can upgrade the work efficiency, competitiveness and even can bring new customers to enterprises (Unhelkar and Murugesan, 2010). Therefore, many enterprises have engaged more people in the research and development of mobile application software to the legacy systems.

The current mainstream platforms for mobile devices such as smartphones and tablet computers are iOS, Android and Windows Phone, but the skills for the implementation of mobile applications in each platform are different. It leads to high cost and requires more labors when enterprises want to develop a mobile application software supporting multi-platforms. Moreover, it will be more difficult for enterprises to develop mobile application servers, not only due to different mobile platforms, but because of many problems in integrating numerous heterogeneous systems. Furthermore, the development and maintenance costs have to be considered along with the modification of legacy systems and the restrictions of mobile devices' hardware and so on. For solving the above problems, building middleware is one of the best solutions.

Middleware makes it easier for software developers to perform communication and input/output, it plays the role as a communication bridge between enterprises' heterogeneous systems and mobile applications and reduces the probability of modifying the legacy systems. The use of middleware can provide the standard encapsulated data format for different mobile platforms so that it can reduce the difficulty of supporting multi-platforms, decrease the duplication of the development efforts and increase the opportunities of the developed components that can be reused.

However, there are no well frameworks can support quickly, development easily and integration conformably to the new mobile application servers in the current mobile middleware. Therefore, for the sake of solving the above problems, this study illustrates the designed mobile middleware framework and has the following characteristics:

- To accomplish in integration the communication between mobile application and legacy system
- To propose a configurable architecture for diverse mobile platforms
- To effectively reduce the demands that modify the legacy system
- To achieve Plugin design for extending middleware functionality

THEORETICAL BACKGROUND

The design framework is mainly related to the study of mobile middleware design and the technology evolution of the integration with legacy systems, the following describes on these two related work.

Technology evolution of the integration with legacy systems: In the recent years, with the rapid development of mobile devices, high demands for supporting mobile devices in existing systems and completion of these tasks have been the major challenges. In order to be able to achieve the above challenges, supporting the Web services in mobile systems is a common way. However, a variety of information systems in the enterprise has been designed and still has to solve various problems in implementation. Therefore, Almonaies *et al.* (2010) illustrates the Service-Oriented Architecture (SOA) evolution collection of system integration and split implement methods as well as strategies into four types:

- **Replacement:** Replacing the legacy system with a new system, this method is also known as the Big-Bang strategy. This strategy can reduce the maintenance costs of the new systems. The disadvantages are time consuming, development expenses and requiring accumulated experiences

- **Wrapping:** This is just support by a Web service interface. The advantage of this strategy is rapid development. This is a good choice for mobile systems which cannot be modified and have high commercial value. The disadvantages are the higher maintenance costs and inefficiency of this method to improve the quality of the legacy systems
- **Redevelopment:** Analyzing the legacy system by using Re-Engineering and Reverse Engineering that modifies the source code and provides a Web service interface. The advantages are the increase in flexibility and reduction of costs. The drawback of this method is the acquirement of source code
- **Migration:** The migration of legacy systems to Web services platforms. The benefit-effort comparisons are between Wrapping and Redevelopment, through retrieving and modifying the source code to provide new Web services interface to co-exist with the existing legacy systems. The advantages are with current tool support and assistant a stable environment. The disadvantages are the time-consumption for the source code development and needs to the accumulation of experience

Almonaies *et al.* (2011) carry on the classification and evolution of the legacy systems and put forward raised other research methods, using Migration strategy to solve the problems of the Web applications migrating to the SOA environment and semi-automated processing migration. The method consists of four-steps:

- Identifying the independent service of Web applications
- To duplicate the service from the independent Web applications
- Directly converting the independent application services into Web services through the SOA library
- To integrate this service with the original Web application

This method can help the current Web applications quickly expand its support for Web services in mobile servers, thus reducing the difficulty in the final design.

To create an interface and avoid the modification of the existing information system for external systems, most enterprises take up the Wrapping strategy. For interactive features of the system, the traditional SOA architecture cannot demonstrate the changes in the interactive process. For that to happen, the external system must understand the interactive process as it access the system supports first. Gerardo Canfora and his research group (Canfora *et al.*, 2008) came up with the implementation of a simulator to communicate with the legacy systems and a description of the XML definition for the legacy systems to respond and for the corresponding state change of an interactive process. Further implementing an automaton engine through which this implementation of the state machine script reaches the interactive communication between the simulator and the legacy system which is provided at the end the SOA interface for the use of external systems. This method provides an effective interactive control and allows the external systems not only know the practice of interaction but also the demand to access to the legacy system resources can be reached.

With the help of tool support, Sneed (2006) provided a method for the rapid development of the Web services interface. The method mainly encapsulate the functionality of legacy systems into the

Web services with the existing SOA tools and automatically generating Web Services Description Language (WSDL) and execute the workflow management through the Business Process Execution Language (BPEL) engine.

Mobile middleware development: Kurogo Mobile Optimized Middleware (Yu, 2012) was developed by Andrew Yu and his research team from the Massachusetts Institute of Technology (MIT). Its main purposes were the middleware development used for mobile applications software and mobile Web, focus on the scalability and clean integration with an excellent users' experience. Kurogo's architecture design includes four parts:

- The Data Source used for converging the models of end data provider and provides for Module usage
- The function modules of Kurogo Server implementation through Module
- HTML Template Engine is used for different mobile devices and through device to detect processing display and to optimize the different mobile Web screen
- In addition, the Web Service provides application function called by native mobile application software

This method offers developers quickly establish support for end use applications and can easily customize the extension's module. However, the limitations of modules are troublesome for customized parts and direct modification to its middleware.

Due to hardware limitations of the mobile device environment and different systems, there are many problems arises in the implementation of a mobile application to supports all platforms. Therefore, Cheung (2005) proposed an environment-aware mobile middleware framework. The main purpose is to adapt the middleware environment for a variety of mobile devices and then assign a more appropriate service for processing. The framework is divided into four modules:

- **Context space:** It contains Context Composer and Detector, parsing a mobile end-side environment with a corresponding combination of related environmental library information
- **Middleware service space:** The main functions contain the public intermediary services and specific application services
- **Adaptation engine:** The main task is based on the requirements of Type of Service (ToS) and Quality of Service (QoS) in a mobile end-side environment with its mobile application to take adaptive actions and assignment of the right service
- **Middleware manager space:** Total operation management in middleware mainly contains five-system management

For enterprises inclined to establish their own mobile business applications provide business workforce with the ability to operate the business data in real time. The biggest challenge is how to extend the mobile function with the legacy system and the need to avoid changing of the existing system code. Most of the current mobile business applications require a lot of customization with increasingly diversified business data and mobile devices. Wei *et al.* (2008) proposed a semantic Web-based integration platform of enterprise information for mobile business that contains four strata:

- **Service level:** The main task is to deal with the action of receiving the request, analyzing and retrieving the essential parameters, then encapsulated the result into a suitable format for the response
- **Semantic level:** It contains a variety of static knowledge of semantic information source components, provides Service level and Mapping level with reasonable semantic information
- **Mapping level:** It includes automatic and manual corresponding ways which existing enterprise application resources with the corresponding description semantics
- **Resources level:** Handling the different commercial data sources in grid style and provides a unified data format for the mobile commerce program

In order to support the rule-based parsing and automatically find the relevant information, these methods provide a handling mechanism for the heterogeneous data in the existing enterprise systems and encapsulate the details of communication with existing enterprise systems, thereby reducing complexity in mobile commerce applications implementation.

MOBILE MIDDLEWARE DESIGN

In order to construct a mobile middleware that can support the rapid development and easy integration of legacy system functions, we proposed mobile middleware architecture (Fig. 1) divided into the following four layers:

- **Router:** To deal with an external program or the client sides from which they can request the corresponding services
- **Plugin framework:** The Plugin architecture features, the mobile middleware that extends to support new functions. The configuration mechanism is used to set them in our design study and make Plugin do the dynamic adjustments. The Plugin can be used more flexibly and the reusability of the Plugin can be improved
- **Common models:** Implements various types of common models, where the developers can easily develop the Plugin and expand to a common model such as customization model which as Videos and News, etc.,
- **Utilities (Tool modules):** Implementation of common elementary tools such as network communications, database access, data format handling, safety-related modules and so on

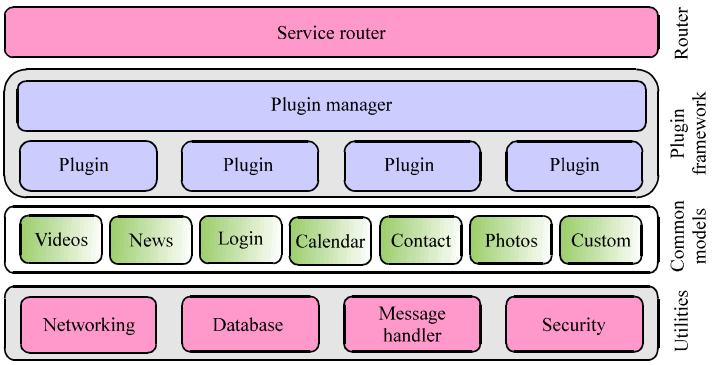

Fig. 1: Configured and extensible mobile middleware architecture

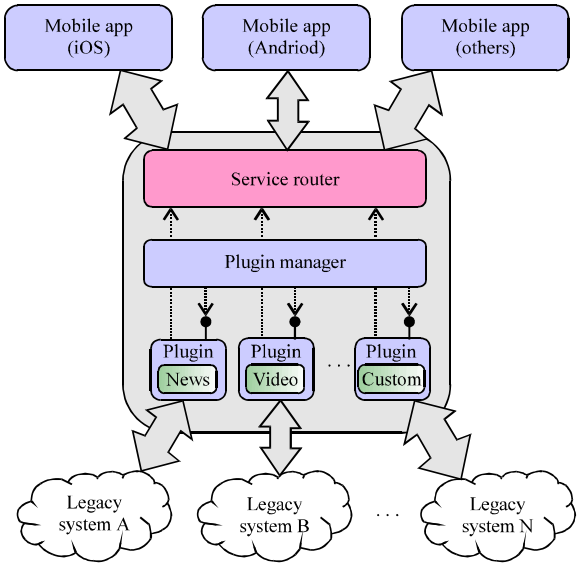

Fig. 2: Uses of plugin type of mobile middleware

Our architecture is designed to minimize the burden on developers. The developers only need to focus on the development of the corresponding Plugin for the legacy system, thereby prompting them to speed up their development efficiency. When developers are looking for developing a generic model for some models which do not support the integration to any legacy systems, it is possible to swiftly self-customize the corresponding models and expand the number of Common Models.

Follow up we will explain the architecture design of mobile middleware and combination with Plugin framework and elucidate the design that mobile application software how to call user interface. Further, we will illustrate how to implement Common Models to access with legacy systems. Finally, we illustrate how to design the Plugin Manager and Plugin.

Mobile middleware framework: Integration of middleware and legacy systems brings a lot of benefits. For example, if heterogeneous mobile systems have same functions, they can share the same middleware components thereby reducing the requirement to modify the legacy systems by middleware and easier to connect with end-side mobile application software of different platforms and so on. However, lesser methods can effectively rapid support the mobile application services of legacy systems leading to developing the mobile servers of manual systems which spends a lot of human and development costs which is not efficient.

To solve the present problems, we add Plugin framework into mobile middleware architecture, let developers only implement Plugin with the integrating systems to complete the supporting mobile application service functions. Figure 2 illustrates our mobile middleware architecture. Service Router is used for connecting with the middleware and assigning the corresponding services to the requests of the mobile end. Plugin Manager is responsible for managing Plugin, checking mounting Plugin and assigning Plugin service object to mount. Finally, Plugin implements the methods that connect with the legacy system functions and use common models to speed up the development of Plugin by the developers.

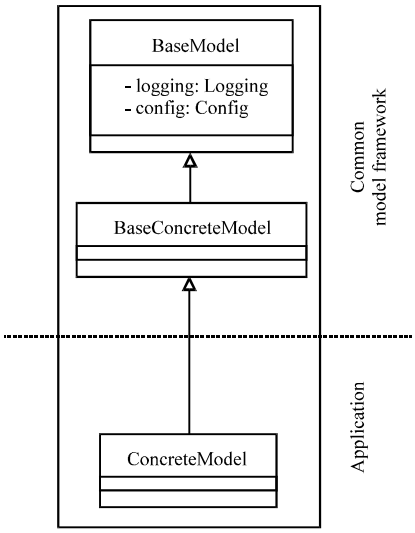

Fig. 3: Model design class diagram

Our mobile middleware was MVP design pattern, which divides the system into Model, View and Presenter. Mobile middleware common models implement on handling the received data and commercial logic. View and Presenter implement a part of Plugin processing and let developers decide these two part designs.

Common model design: A number of mobile systems exist and an enterprise may consist of a variety of different legacy systems. However, most of these similar systems are likely to have similar or same functionalities. To be able to reduce the duplication of development components and quickly create a new model that supports more system designs, a Common Model Framework in this study is proposed for different types of functions and builds different classification abstract model that allow developers to implement various types of functions.

Figure 3 illustrates a common model framework for the design of this study. Various types of abstract models inherit the BaseModel parent class. It has many common model features, such as logging, configuring etc. These various types of functional models are implemented in their own methods according to their features types which are commonly used in the development of specific models; therefore, being able to finalize on the specific model is to be used to provide the development Plugin.

Configuration Plugin administrator design: The Framework PluginManager plays a very important role in its functions for the management and generation of Plugin components. When you want to use the Plugin component at runtime, you can dynamically find and generate the corresponding Plugin components. In order to manage centrally the configuration of Plugin, through the configuration assigned to the Plugin mount services and support service functions of middleware are configurable. In addition to the common Plugin administrators design methods for the design of the PluginManager, the mobile middleware software is configurable. The PluginManager is designed for connecting with the configurable features so that the PluginManager has the ability to read the configuration contents, to manage the relationships of Plugin and services and be able to adjust dynamically with the mounted Plugin services.

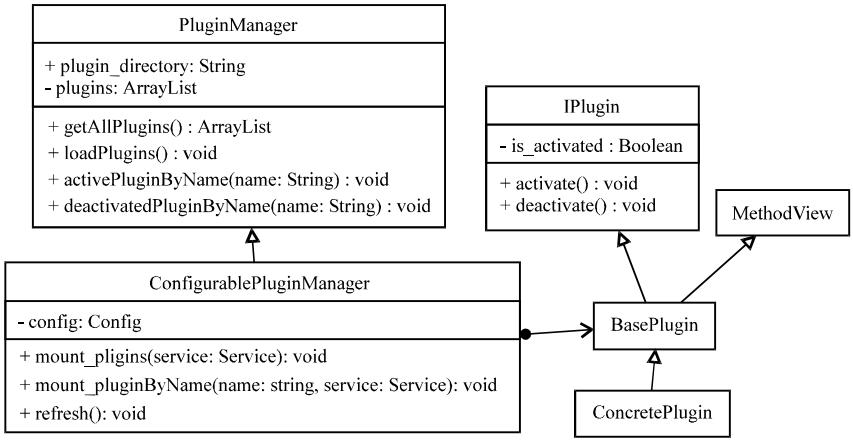

Fig. 4: PluginManager class diagram

```
---

# Plugin名稱:
#    uri: 對應服務的URI
#    methods: 可使用的http方法清單
#    is_activated: 是否掛載

Plugin1:
    uri: /plugin1
    methods: [GET]
    is_activated: true
Plugin2:
    uri: /plugin2
    methods: [GET,POST]
    is_activated: false

...
```

Fig. 5: Format of the configuration file

Figure 4 shows how it can be configured with the PluginManager. The components of the design's main content contain the directory location of the Plugin, Plugin list and get method, to load Plugin method, mount and unload Plugin method; ConfigurablePluginManager follows the class by the inheritance of PluginManager's parent class, implement the configuration management and assign the Plugin mount service.

The ConfigurablePluginManager in Fig. 5 displays configuration formats, the main use of the YAML language to design configuration format. The contents of the configuration corresponding to the name of the Plugin are set, the configurable Plugin parameter corresponds to service URI, the list of the HTTP methods used in and whether activated.

Pattern design: The design allows the service to use the Plugin which in our study shows Plugin function and inherit MethodView class. The main function of the class is to achieve the method of HTTP request. Different uses of functions (add, get, update and remove) are developed by Template

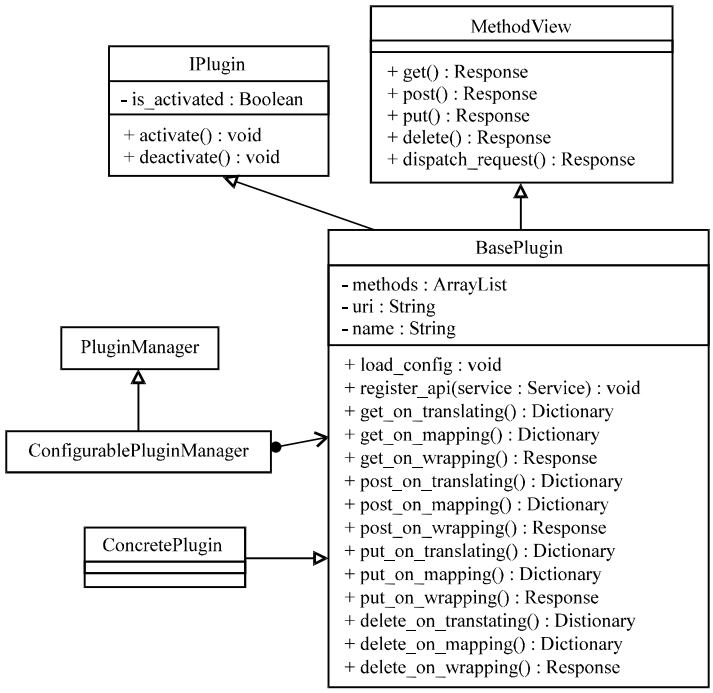

Fig. 6: Plugin design class diagram

Pattern deployment which requires the developers to complete the corresponding analytical purposes. The correspondence and encapsulation method is used to complete the integration of legacy systems enabled services and join function of reading configuration file, allowing developers through dynamic parameter adjustment to reuse the Plugin.

In Fig. 6 illustrates the Plugin design, the BasePlugin abstract class inherits IPlugin class and MethodView class to achieve Plugin functions and HTTP request method respectively. Loading configuration functions (load_config) allows developers to adjust dynamically the parameters. Registration service feature (register_api) makes Plugin be able to implement Web services. HTTP four actions are designed to resolve the corresponding and encapsulating abstract method that allows the developers to achieve the functional use. For parsing and encapsulation stage, due to the large common functions that are the same, we implement two abstract methods in the BasePlugin abstract class for parsing retrieved HTTP parameters and encapsulating JSON data format.

After accepting the request of mobile-end, Plugin instance that uses this process which is shown in Fig. 7. The mobile application software issues a HTTP request depending on the action first Plugin receives to perform the corresponding analytical method (on_translating), parsing HTTP request parameters. Subsequently, the corresponding method (on_mapping) taking over the tasks is requested and the corresponding model is called to deal with the legacy systems to communicate and accesses information. The last call encapsulation method (on_wrapping) along with results will be packed into specific data formats such as JSON, XML and so on and then is added to the HTTP response back to the mobile application software to complete the integrated mobile application services of legacy system.

Summarizing our approach, whenever developers need to develop integration on mobile systems, developers must first confirm whether the mobile middleware enables the support for the common models. If there is a supported model, developers with the Plugin only need to complete the corresponding function analysis, correspondence and encapsulation method. Otherwise, you will need to create a common model to the corresponding function and then develop the Plugin through the configuration file in order to control the corresponding URL and finally parameter settings have to be completed.

CASE STUDY

We applied this mobile architecture in the FCU mobile application software middleware (FCUMW), Android and iOS platform for FCU mobile application software usage. We took one of the function Campus Messages for an example (Fig. 8), illustrating how to implement a Mobile App integrating system.

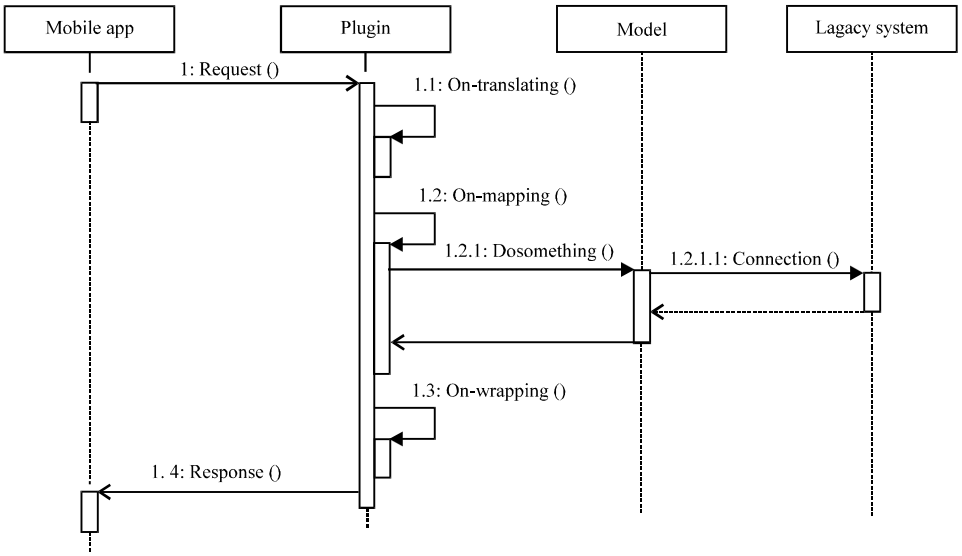

Fig. 7: Plugin sequence diagram

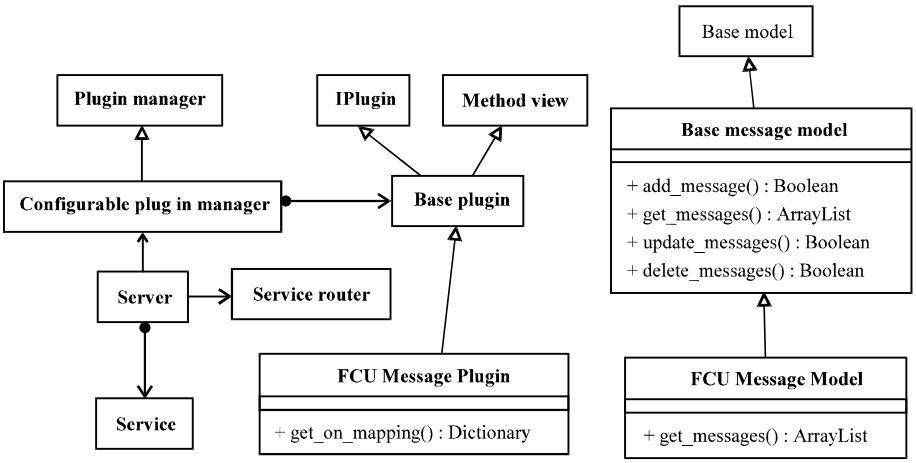

Fig. 8: Campus message Plugin class diagram

Message abstract model implements the BaseMessageModel abstract class that defines four methods to achieve the following add, get, update and remove messages. We have inherited message abstract class, the real FCUMessageModel class implementation to acquire the message function. In the case of campus message Plugin, we inherit BasePlugin abstract class to implement FCUMessagePlugin class only in implementing class for get_on_mapping function.

FCUMW-campus message: Since the beginning of mobile middleware, it has not established the common model for school message. Therefore, we inherited the message model to implement FCUMessageModel class, as the following code fragment:

```
class FCUMessageModel (BaseNessageNodel):
 def_init_(self, logger = None):
  BaseMessageModel._init_(self, logger)
 def get_message (self, *args, **kwargs):
  start_date = kwargs['start_date']
  end_date = kwargs['start_date']
  ...
  messages = self.session.query(message)\
    .filter(Message.expire_date>date>end_date)

.filter(Message.publish_date>date>end_date)
```

FCUMessageModel class only implements on tracking campus messages. According to the different conditions of the different parameter settings, the results of a database query returns (return massage) are performed in the final implementation stage. In addition, if you need to add, update and delete messages and do other functions, just implement the corresponding method.

Implementation of Plugin-campus message: The Campus Message example only implements to acquire the message function and uses default resolution and encapsulation method. So the FCUMessagePlugin need only be completed using the get_on_mapping function implementation, as in the following code fragment. FCUMessageModel message is called and then return the result to complete the task.

```
class FCUMessagePlugin(BasePlugin):
 def get_on_mapping(self, **kwargs):
 try:
 check_api_key()
 model = FCUMessageModel()
          m e s s a g e s          =
model.get_messages(kwargs)
 count = messages.count()
 return dict(data=data, count=count)
 except MappingHandlerError:
```

RESULTS AND DISCUSSION

The description of the above two subsections and development of Plugin actually just completes get_on_mapping function and the FCUMessageModel class. The other features implementation such as the above implementation can significantly reduce the burden on the developer.

A configurable and extensible mobile middleware based on web service and plugin architecture to make the integration easier and achieve extensibility. To reduce the load of modifying programs, we abstract the management of plugin into a configuration file, such that we just modify the configuration file when adjusting the plugin, instead of revising the programs.

CONCLUSION

This study provides the mobile middleware design architecture to solve the development and integration of the legacy system functions with mobile application service in the enterprises. By implementing design framework in the common model reduces the duplication of development costs. The developer just focuses on the design of the Plugin so that it will speed up and reduce the difficulties during the current system development or system extend functions. The characteristics that can be configured and Plugin framework allows developers to adjust dynamically the Plugin Manager and Plugin, making the Plugin more reusable.

This design architecture is applied to the mobile FCUMW applications. The architecture is proved along with results to achieve the following conclusions:

- A standardized protocol and its data format in the mobile platform are achieved
- Mobile middleware which can be dynamically adjusted
- The rapid development for the expansion of mobile application services is realized
- Reduction of the needs to modify the legacy systems
- Easy to develop Plugin design

In the future, we hope to add more features such as service workflow, service matchmaking and service composition designed for our mobile middleware along with the settings that can produce Plugin and implement more protocols and data formats.

ACKNOWLEDGMENTS

This research was sponsored by the National Science Council of Taiwan R.O.C. under the Grant NSC101-2221-E-035-027. We would also like to thank deeply the anonymous reviewers for their valuable comments and feedbacks.

REFERENCES

Almonaies, A., M. H. Alalfi, J. R. Cordy and T. R. Dean, 2011. Towards a framework for migrating web applications to web services. Proceedings of the Conference of the Center for Advanced Studies on Collaborative Research, November 7-10, 2011, Toronto, Ontario, Canada, pp: 229-241.

Almonaies, A.A., J.R. Cordy and T.R. Dean, 2010. Legacy system evolution towards service-oriented architecture. Proceedings of the International Workshop on SOA Migration and Evolution, March 15-18, 2010, Spain, pp: 53-62.

Canfora, G., A.R. Fasolino, G. Frattolillo and P. Tramontana, 2008. A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. J. Syst. Software, 81: 463-480.

Cheung, R., 2005. An adaptive middleware infrastructure for mobile computing. Proceedings of the Special Interest Tracks and Posters of the 14th International Conference on World Wide Web, May 10-14, 2005, Chiba, Japan, pp: 996-997.

Sneed, H., 2006. Integrating legacy software into a service oriented architecture. Proceedings of the 10th European Conference on Software Maintenance and Reengineering, March 22-24, 2006, Bari, Italy, pp: 11-14.

Unhelkar, B. and S. Murugesan, 2010. The enterprise mobile applications development framework. IT Professional, 12: 33-39.

Wauters, R., 2012. IDC predicts 183 billion mobile app downloads by 2015, rise of in-app purchasing. International Data Corporation.

Wei, Z., M. Kang and W. Zhou, 2008. A semantic Web-based enterprise information integration platform for mobile commerce. Proceedings of the International Conference on Management of E-Commerce and E-Government, October 17-19, 2008, Jiangxi, China, pp: 57-60.

Yu, A., 2012. The Kurogo server: Mobile optimized middleware. The Technology Behind Kurogo, http://kurogo.org/technology/.