Package Delivery Application System

Zeng Jiaxing 1452680 | Li Yichao 1452666 | Liu Lidong 1354362 | Yi Siqi 1452662

June 10, 2016

Table of Contents

1 OverView

1.1 Progress

Based on analysis models we designed in last several assignments, our system is put into practice this time. Several progress has been made to make our program more feasible and reliant.

We firstly consider architecture refinement according to platform and frameworks we will use. Some layers are merged with others, while new ones are added as necessary. In this part, we not only pay attention to the progress of realizing the system, but take system security, persistence, data storage and so on into account.

As high-level architecture is still a conceptual design, we then give more details of subsystems on service layer, including payment, order, delivery userInfo, route planning and message subsystems. A list of interfaces is offered to show how our subsystems work and their responsibilities as well as dependencies on others.

To demonstrate interface specification in details, we draw a class diagram of payment subsystem, which is responsible for order payment business. Several outer third-party APIs, Alipay for example, are also included in this diagram. Its realization is shown in our prototyping.

Considering design difficulties and feasibilities, we choose two analysis mechanisms, persistence and message routing, to implement. Unlike analysis mechanisms, design mechanisms are practical and platform-dependent. We spend a large amount of time on this part and do learn a lot during the process of searching relative frameworks and techniques, as there are so many solutions but few practical experiences we have.

Moreover, we update our use case realizations diagrams. Comparing to the old ones, components are now replaced by platform-dependent classes and subsystem interfaces. All boundaries are divided into one class and relating subsystems, while controllers have more specific functions: servlet action classes as overall controller and action classes doing detailed control.

Finally, we do a prototype, which contains a small part of our project, as an examination of platform and frameworks we choose. The result works as we expected, although much more should be done to build a whole system.

We have learned more than just designing but practicing this time and honestly speaking, that is beyond our anticipations when we first chose this course. A challenge really can expand one' s horizons in such a short time.

1.2 Platform & frameworks

Overall, our system is constructed on Java EE, a platform which provides an API and runtime environment for developing and running enterprise software, including network and web services, and other large-scale, multi-tiered, scalable, reliable, and secure network applications.

Frameworks used for each layer are list as below, they are all mature technique and offer ideal solutions to problems we might meet during implementation.

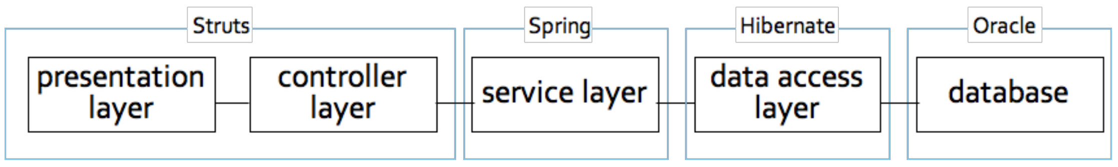


Figure 1.1 overall frameworks

For presentation and controller layer, we choose Apache Struts 2, an open-source web application framework for developing Java EE web applications. It uses and extends the Java Servlet API to adopt a model - view - controller (MVC) architecture, which is exactly used in our project. Another advantage of using Struts is its easy extension with plugins for Spring and Hibernate, frameworks we select for other two layers.

For service layer, we choose Spring as the main maintainer. The Spring Framework is an open source application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform. It acts as an addition to the Enterprise JavaBeans (EJB) model. Another reason we choose Spring is its good compatibility with Struts and Hibernate, which is vital for our system coordination.

As for data access layer, due to its far-reaching influence and popularity, Hibernate is our first choice. Hibernate ORM is an free object-relational mapping framework for the Java language. It provides a framework for mapping an object-oriented domain model to a relational database. Its primary feature is mapping from Java classes to database tables and mapping from Java data types to SQL data types. Thanks to Hibernate, we can solve persistence problem in a convenient and safe way.

Finally is Oracle Database, a well known object-relational database management system produced and marketed by Oracle Corporation. Obviously it has strong productivity and satisfies our system’s needs for database.

More details of each layer are set out as below.

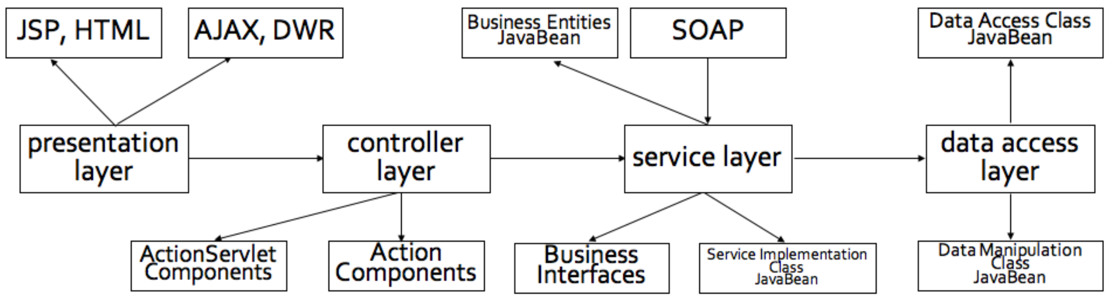


Figure 1.2 details in each framework

On presentation layer, JSP language is used on server side, while HTML is on client side. DWR framework is used for two main reasons: allow JavaScript to retrieve data from a servlet-based web server using Ajax principles; make it easier to dynamically update the web page with the retrieved data. On controller layer, we have ActionServlet Components, serving as core controllers; action components, responsible for detailed control works. They are both kernel parts in Struts.

On service layer, we provide business interfaces as access to different services. Service implementation classes and business entities, are indispensable JavaBeans classes in service logic. To realize remote access functions, we apply SOAP protocols and RMI as an supplement to Spring's Remote Access framework. On data access layer, we design data access classes and data manipulation classes, both are JavaBeans.

1.3 Deployment view

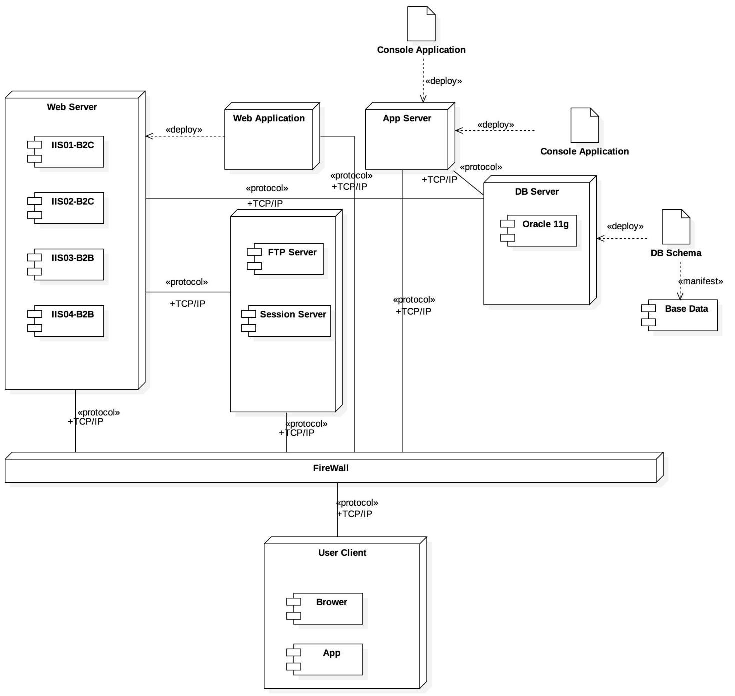


Figure 1.3 deployment diagram

2 Architecture Refinement

a. Platform-dependent architecture

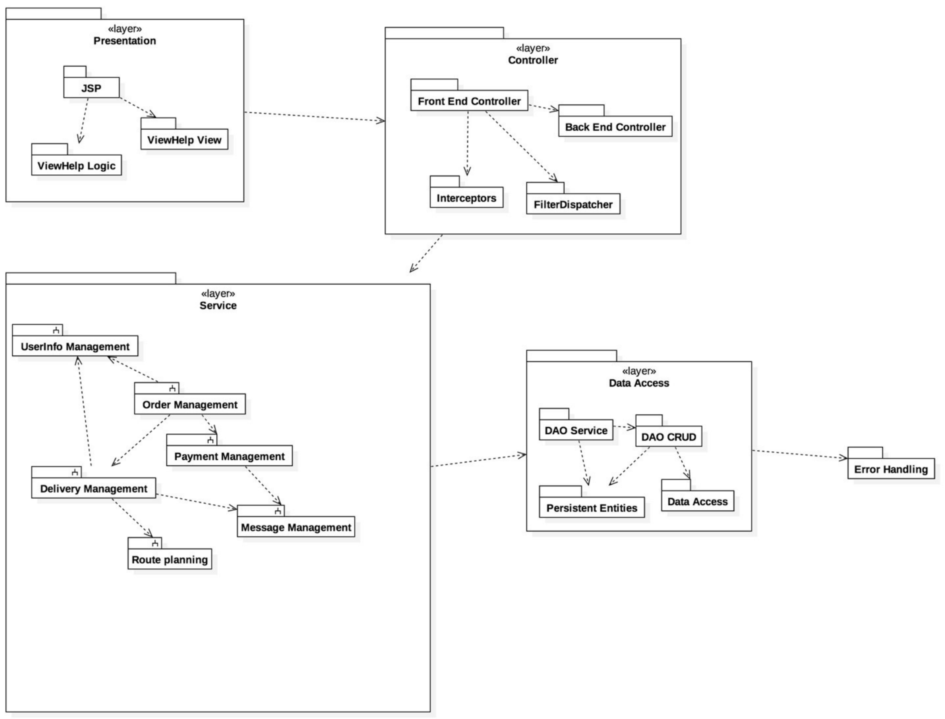


Figure 2.a platform-dependent architecture

b. List of subsystems and interfaces

Subsystem	Interface

Payment	**Provided Interface:**
	IPayment: createPayment()
	cancelPayment()
	payByOrderID()
	close()
	refund()

	Required Interface:
	alipay:alipay.trade.create()
	alipay.trade.cancel()
	alipay.trade.pay()
	alipay.trade.close()
	alipay.trade.refund()
	alipay.trade.query()
	UnionPay:unionpay.trade.create()
	unionpay.trade.cancel()
	unionpay.trade.pay()
	unionpay.trade.close()
	unionpay.trade.refund()
	unionpay.trade.query()

Order	**Provided Interface:**
	IOder:createOrder()
	updateStatement()
	searchOrder()
	cancelOrder()

	Required Interface:
	IDelivery:createDeliveryPlan()
	updateStatement()
	searchCurrentLocation()
	receiveDelivery()

Delivery	**Provided Interface:**
	IDelivery:createDeliveryPlan()
	updateStatement()
	searchCurrentLocation()
	receiveDelivery()

	Required Interface:
	gaodeMap:AMap.Transfer()
	AMap.Walking()
	AMap.Driving()
	AMap.Geocoder.getAddress()
	AMap.Geocoder.getLocation()
	KeychainTouchID:
	Signature:

UserInfo	**Provided Interface:**
	ISet:createAccount()
	updateAccount()
	verifyIdentification()
	deleteAccount()
	getAccountInfo()

Subsystem	Interface

	Required Interface:
	gaodeMap:AMap.Transfer()
	AMap.Walking()
	AMap.Driving()
	AMap.Geocoder.getAddress()
	AMap.Geocoder.getLocation()

c&d. Detailed Subsystem & Interface Specification

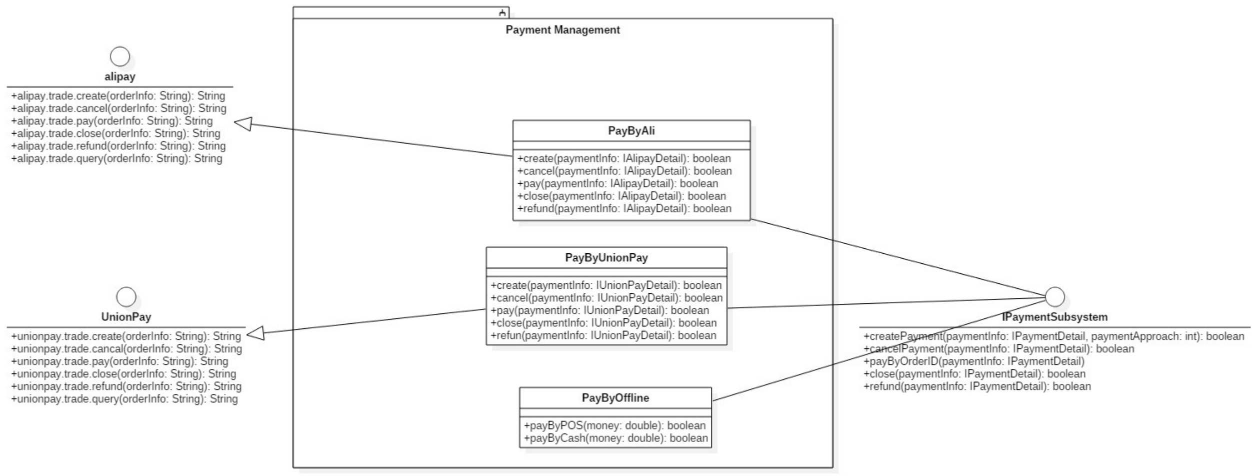

Payment Management
alipay
+alipay.trade.create(orderInfo: String): String
+alipay.trade.cancel(orderInfo: String): String
+alipay.trade.pay(orderInfo: String): String
+alipay.trade.close(orderInfo: String): String
+alipay.trade.refund(orderInfo: String): String
+alipay.trade.query(orderInfo: String): String
PayByAli
+create(paymentInfo: IAlipayDetail): boolean
+cancel(paymentInfo: IAlipayDetail): boolean
+pay(paymentInfo: IAlipayDetail): boolean
+close(paymentInfo: IAlipayDetail): boolean
+refund(paymentInfo: IAlipayDetail): boolean
PayByUnionPay
+create(paymentInfo: IUnionPayDetail): boolean
+cancel(paymentInfo: IUnionPayDetail): boolean
+pay(paymentInfo: IUnionPayDetail): boolean
+close(paymentInfo: IUnionPayDetail): boolean
+refun(paymentInfo: IUnionPayDetail): boolean
UnionPay
+unionpay.trade.create(orderInfo: String): String
+unionpay.trade.cancal(orderInfo: String): String
+unionpay.trade.pay(orderInfo: String): String
+unionpay.trade.close(orderInfo: String): String
+unionpay.trade.refund(orderInfo: String): String
+unionpay.trade.query(orderInfo: String): String
PayByOffline
+payByPOS(money: double): boolean
+payByCash(money: double): boolean
IPaymentSubsystem
+createPayment(paymentInfo: IPaymentDetail, paymentApproach: int): boolean
+cancelPayment(paymentInfo: IPaymentDetail): boolean
+payByOrderID(paymentInfo: IPaymentDetail)
+close(paymentInfo: IPaymentDetail): boolean
+refund(paymentInfo: IPaymentDetail): boolean

Figure 2.c&d payment subsystem and interface specification

We choose Payment Subsystem as an example to specify its inner and external interface in details to demonstrate our refined architecture. The Payment Subsystem is designed for realizing purchasing function of the whole model. We considered several main purchase patterns in real life, such as online payment like Alipay which is becoming popular nowadays, the traditional payment method —Unionpay which is suitable for large amount of money and of course the basic pattern—paying by cash, for we should consider senior citizen who are not familiar with other payment pattern. To use all the third-party payment patterns mentioned above, we use the adaptor model: decorating the interface provides by the Ali and Union. Then the classes inside the subsystem are able to use these interface without mismatching. The payment provides the interface IPayment to other subsystem and parts of out delivery system with methods to create, cancel and do the payment.

3 Design Mechanism

3.1 Persistence

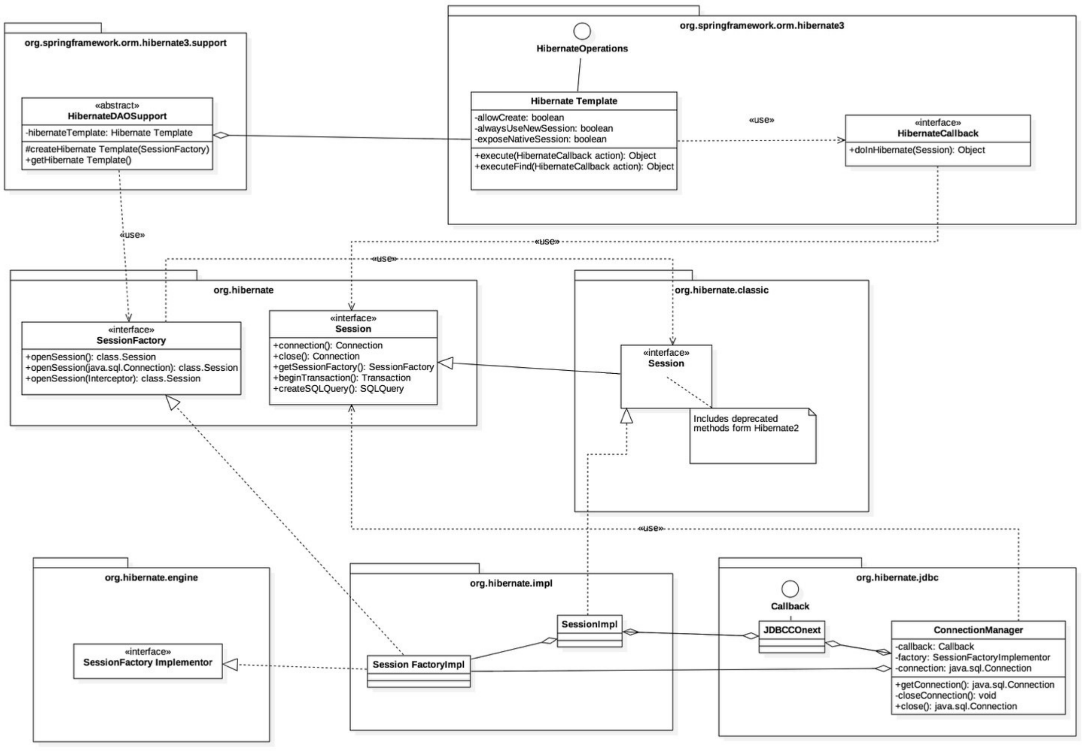

org.springframework.orm.hibernate3.support
«abstract» HibernateDAOSupport
-hibernateTemplate: Hibernate Template
#createHibernate Template(SessionFactory)
+getHibernate Template()
org.springframework.orm.hibernate3
HibernateOperations
Hibernate Template
-allowCreate: boolean
-alwaysUseNewSession: boolean
-exposeNativeSession: boolean
+execute(HibernateCallback action): Object
+executeFind(HibernateCallback action): Object
«use»
«interface» HibernateCallback
+doInHibernate(Session): Object
org.hibernate
«interface» SessionFactory
+openSession(): class.Session
+openSession(java.sql.Connection): class.Session
+openSession(Interceptor): class.Session
«interface» Session
+connection(): Connection
+close(): Connection
+getSessionFactory(): SessionFactory
+beginTransaction(): Transaction
+createSQLQuery(): SQLQuery
org.hibernate.classic
«interface» Session
Includes deprecated methods form Hibernate2
org.hibernate.engine
«interface» SessionFactory Implementor
org.hibernate.impl
SessionImpl
Session FactoryImpl
org.hibernate.jdbc
Callback
JDBCCOnext
ConnectionManager
-callback: Callback
-factory: SessionFactoryImplementor
-connection: java.sql.Connection
+getConnection(): java.sql.Connection
-closeConnection(): void
+close(): java.sql.Connection

Figure 3.1 persistence mechanism

The biggest concern in persistence mechanism is how our system takes the values from Java class attributes and persists them to a database table. The diagram above gives a brief description of the whole process.

As mentioned before, our system takes advantage of MVC architecture, so data storage happens between data access layer and database, and the transaction from object class to database mainly depends on OR(object relation) Mapping framework. In this system, we apply Hibernate.

As shown on the first line, Hibernate gets support from Spring framework (org. springframework. orm. hibernate3. support, org. springframework. orm. hibernate3), which helps transfer data from Spring to Hibernate. Then we will use Session interface, the main interaction interface between Java class and Hibernate. We also need SessionFactory, whose responsibility is to create Session instances. Usually an application has a single SessionFactory instance and threads servicing client requests obtain Session instances from this factory. Finally we has org.hibernate.jdbc(this package is shown at the right lower corner), which includes a ConnectionManager to connect Hibernate with JDBC.

And lower level business, such as database connection, is actually done by JDBC, we do not concern these details as Hibernate already encapsulates them well.

3.2 Message Routing

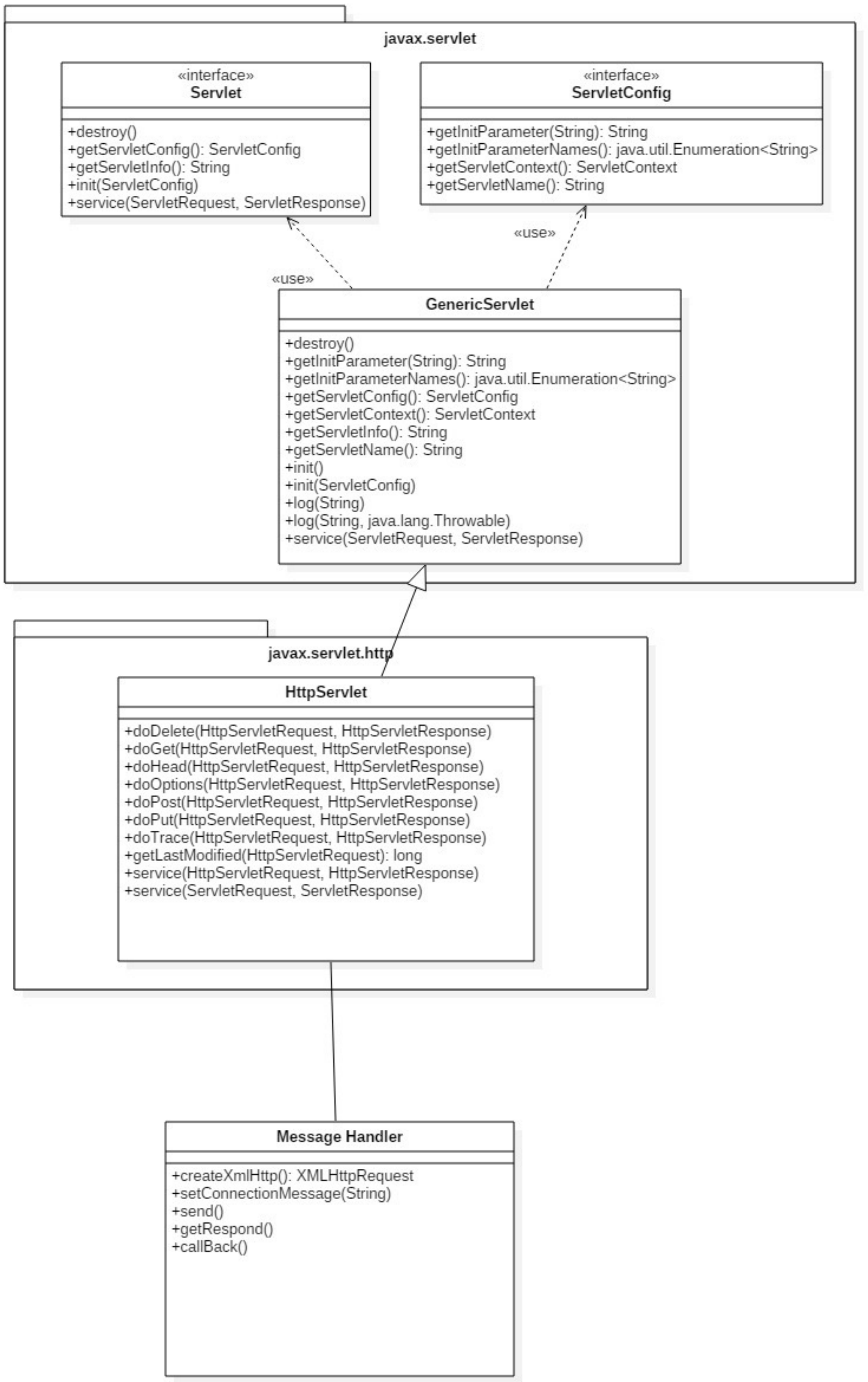

javax.servlet
«interface»
Servlet
+destroy()
+getServletConfig(): ServletConfig
+getServletInfo(): String
+init(ServletConfig)
+service(ServletRequest, ServletResponse)
«interface»
ServletConfig
+getInitParameter(String): String
+getInitParameterNames(): java.util.Enumeration<String>
+getServletContext(): ServletContext
+getServletName(): String
«use»
«use»
GenericServlet
+destroy()
+getInitParameter(String): String
+getInitParameterNames(): java.util.Enumeration<String>
+getServletConfig(): ServletConfig
+getServletContext(): ServletContext
+getServletInfo(): String
+getServletName(): String
+init()
+init(ServletConfig)
+log(String)
+log(String, java.lang.Throwable)
+service(ServletRequest, ServletResponse)
javax.servlet.http
HttpServlet
+doDelete(HttpServletRequest, HttpServletResponse)
+doGet(HttpServletRequest, HttpServletResponse)
+doHead(HttpServletRequest, HttpServletResponse)
+doOptions(HttpServletRequest, HttpServletResponse)
+doPost(HttpServletRequest, HttpServletResponse)
+doPut(HttpServletRequest, HttpServletResponse)
+doTrace(HttpServletRequest, HttpServletResponse)
+getLastModified(HttpServletRequest): long
+service(HttpServletRequest, HttpServletResponse)
+service(ServletRequest, ServletResponse)
Message Handler
+createXmlHttp(): XMLHttpRequest
+setConnectionMessage(String)
+send()
+getRespond()
+callBack()

Figure 3.2 message routing mechanism

The diagram above gives a brief description of the structure.

The Message Handler of application in browser will keep sending Ajax request to the web server and get responds. The Ajax request includes the server URL and customer ID.

The HttpServlet of web server will respond the request with a string which contains the customer ID and other information. The responding string will also contain a message if there is a message need to be sent to the customer. HttpServlet will implement two interfaces. The Servlet interface will contain the method to send message while the ServletConfig interface will provide the configuration of sending message.

4 Use Case Realization

4.1 PayByAlipay

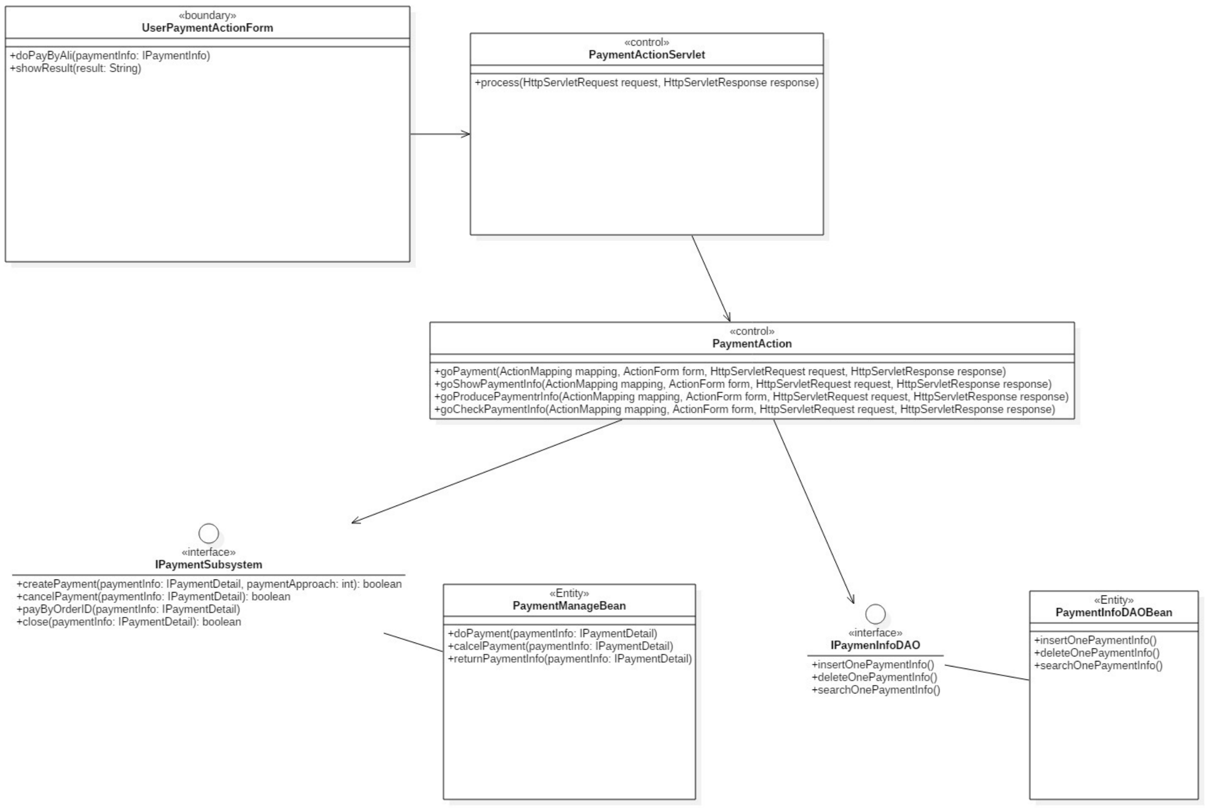

«boundary»
UserPaymentActionForm
+doPayByAli(paymentInfo: IPaymentInfo)
+showResult(result: String)
«control»
PaymentActionServlet
+process(HttpServletRequest request, HttpServletResponse response)
«control»
PaymentAction
+goPayment(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)
+goShowPaymentInfo(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)
+goProducePaymentrInfo(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)
+goCheckPaymentInfo(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)
«interface»
IPaymentSubsystem
+createPayment(paymentInfo: IPaymentDetail, paymentApproach: int): boolean
+cancelPayment(paymentInfo: IPaymentDetail): boolean
+payByOrderID(paymentInfo: IPaymentDetail)
+close(paymentInfo: IPaymentDetail): boolean
«Entity»
PaymentManageBean
+doPayment(paymentInfo: IPaymentDetail)
+calcelPayment(paymentInfo: IPaymentDetail)
+returnPaymentInfo(paymentInfo: IPaymentDetail)
«interface»
IPaymenInfoDAO
+insertOnePaymentInfo()
+deleteOnePaymentInfo()
+searchOnePaymentInfo()
«Entity»
PaymentInfoDAOBean
+insertOnePaymentInfo()
+deleteOnePaymentInfo()
+searchOnePaymentInfo()

Figure 4.1.1 pay by ali class diagram

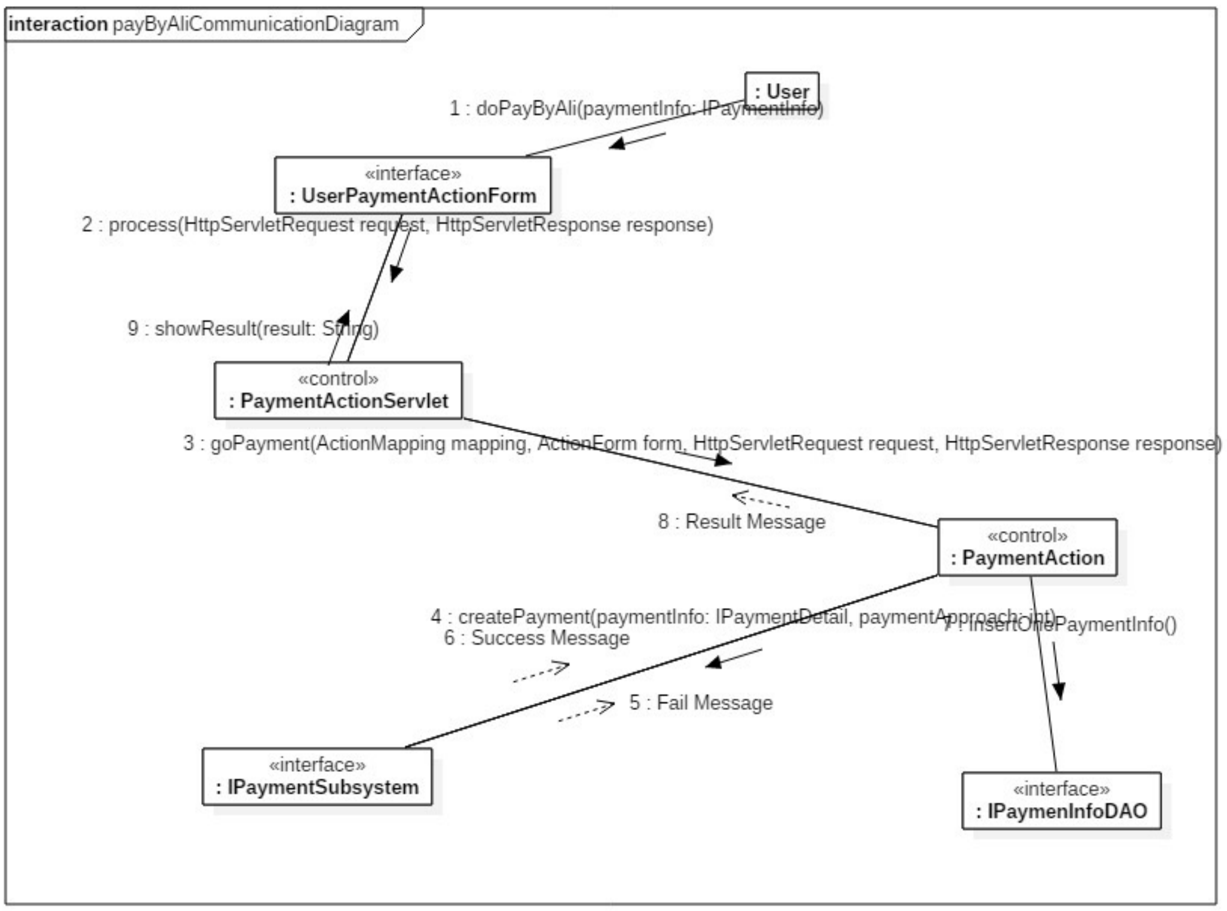

interaction payByAliCommunicationDiagram
: User
1 : doPayByAli(paymentInfo: IPaymentInfo)
«interface»
: UserPaymentActionForm
2 : process(HttpServletRequest request, HttpServletResponse response)
9 : showResult(result: String)
«control»
: PaymentActionServlet
3 : goPayment(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)
8 : Result Message
«control»
: PaymentAction
4 : createPayment(paymentInfo: IPaymentDetail, paymentApproach: int)
6 : Success Message
7 : insertOnePaymentInfo()
5 : Fail Message
«interface»
: IPaymentSubsystem
«interface»
: IPaymenInfoDAO

Figure 4.1.2 pay by ali communication diagram

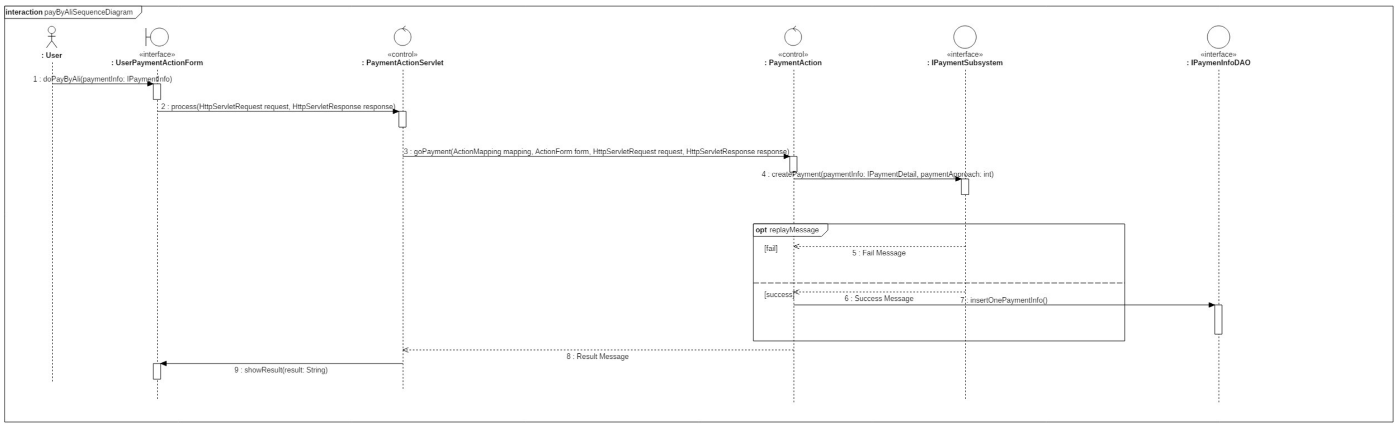

interaction payByAliSequenceDiagram
: User
«interface»
: UserPaymentActionForm
«control»
: PaymentActionServlet
«control»
: PaymentAction
«interface»
: IPaymentSubsystem
«interface»
: IPaymenInfoDAO
1 : doPayByAli(paymentInfo: IPaymentInfo)
2 : process(HttpServletRequest request, HttpServletResponse response)
3 : goPayment(ActionMapping mapping, ActionForm form, HttpServletRequest request, HttpServletResponse response)
4 : createPayment(paymentInfo: IPaymentDetail, paymentApproach: int)
opt replayMessage
[fail]
5 : Fail Message
[success]
6 : Success Message
7 : insertOnePaymentInfo()
8 : Result Message
9 : showResult(result: String)

Figure 4.1.3 pay by ali sequence diagram

4.2 Message Routing

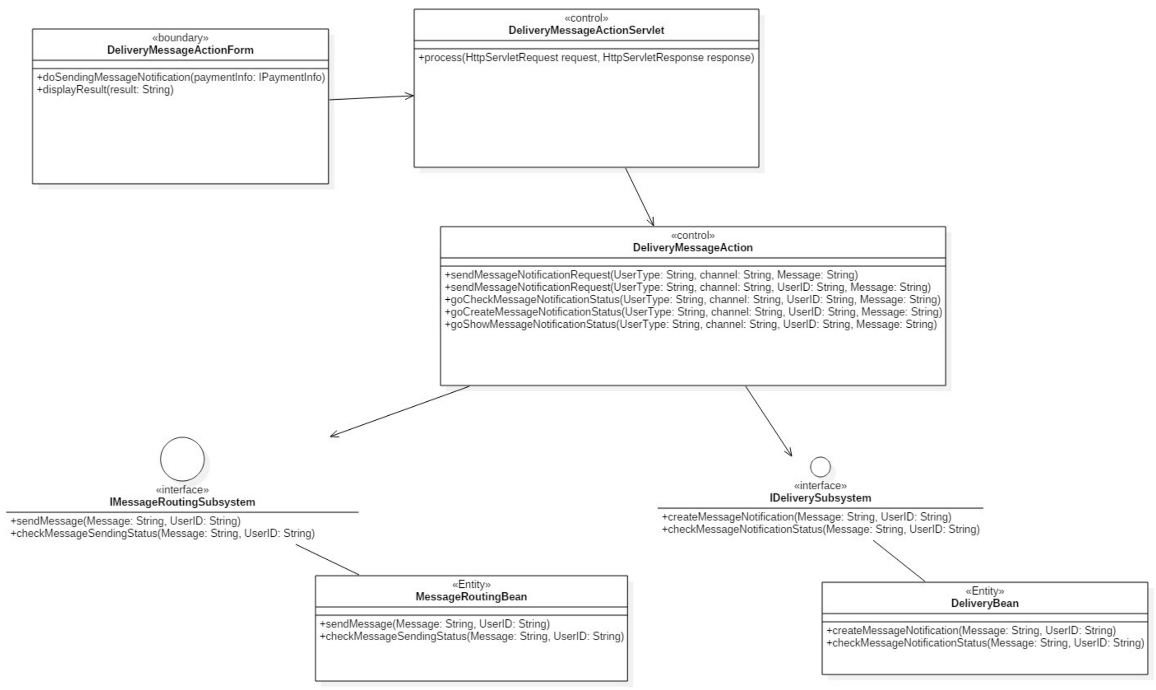

«boundary»
DeliveryMessageActionForm
+doSendingMessageNotification(paymentInfo: IPaymentInfo)
+displayResult(result: String)
«control»
DeliveryMessageActionServlet
+process(HttpServletRequest request, HttpServletResponse response)
«control»
DeliveryMessageAction
+sendMessageNotificationRequest(UserType: String, channel: String, Message: String)
+sendMessageNotificationRequest(UserType: String, channel: String, UserID: String, Message: String)
+goCheckMessageNotificationStatus(UserType: String, channel: String, UserID: String, Message: String)
+goCreateMessageNotificationStatus(UserType: String, channel: String, UserID: String, Message: String)
+goShowMessageNotificationStatus(UserType: String, channel: String, UserID: String, Message: String)
«interface»
IMessageRoutingSubsystem
+sendMessage(Message: String, UserID: String)
+checkMessageSendingStatus(Message: String, UserID: String)
«interface»
IDeliverySubsystem
+createMessageNotification(Message: String, UserID: String)
+checkMessageNotificationStatus(Message: String, UserID: String)
«Entity»
MessageRoutingBean
+sendMessage(Message: String, UserID: String)
+checkMessageSendingStatus(Message: String, UserID: String)
«Entity»
DeliveryBean
+createMessageNotification(Message: String, UserID: String)
+checkMessageNotificationStatus(Message: String, UserID: String)

Figure 4.2.1 message routing class diagram

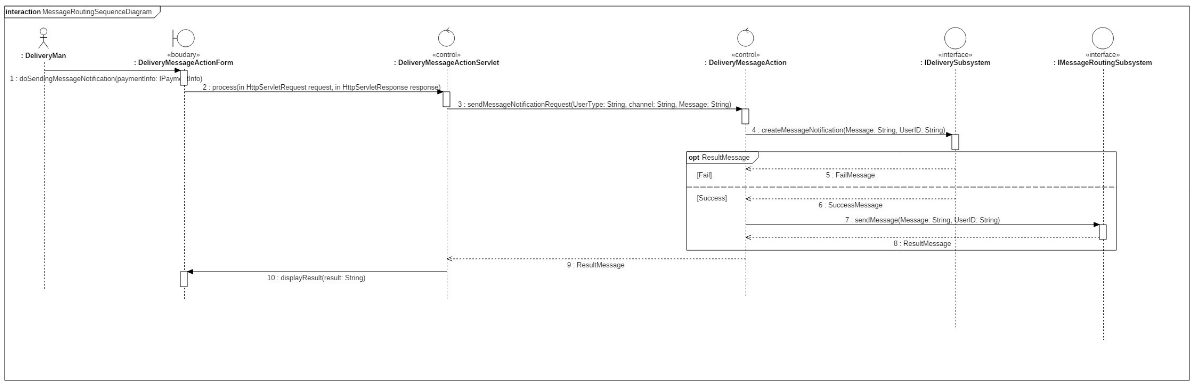

interaction MessageRoutingSequenceDiagram
: DeliveryMan
«boudary»
: DeliveryMessageActionForm
«control»
: DeliveryMessageActionServlet
«control»
: DeliveryMessageAction
«interface»
: IDeliverySubsystem
«interface»
: IMessageRoutingSubsystem
1 : doSendingMessageNotification(paymentInfo: IPaymentInfo)
2 : process(in HttpServletRequest request, in HttpServletResponse response)
3 : sendMessageNotificationRequest(UserType: String, channel: String, Message: String)
4 : createMessageNotification(Message: String, UserID: String)
opt ResultMessage
[Fail]
5 : FailMessage
[Success]
6 : SuccessMessage
7 : sendMessage(Message: String, UserID: String)
8 : ResultMessage
9 : ResultMessage
10 : displayResult(result: String)

Figure 4.2.2 message routing sequence diagram

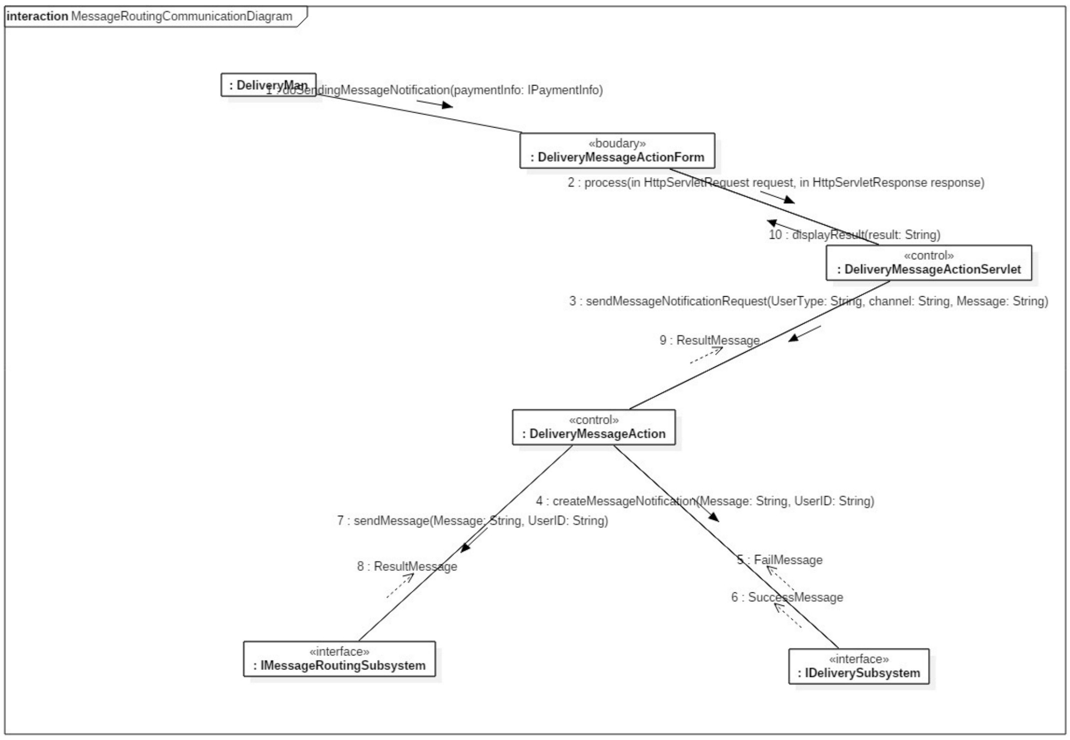

interaction MessageRoutingCommunicationDiagram
: DeliveryMan
1 : doSendingMessageNotification(paymentInfo: IPaymentInfo)
«boudary»
: DeliveryMessageActionForm
2 : process(in HttpServletRequest request, in HttpServletResponse response)
10 : displayResult(result: String)
«control»
: DeliveryMessageActionServlet
3 : sendMessageNotificationRequest(UserType: String, channel: String, Message: String)
9 : ResultMessage
«control»
: DeliveryMessageAction
4 : createMessageNotification(Message: String, UserID: String)
7 : sendMessage(Message: String, UserID: String)
8 : ResultMessage
5 : FailMessage
6 : SuccessMessage
«interface»
: IMessageRoutingSubsystem
«interface»
: IDeliverySubsystem

Figure 4.2.3 message routing communication diagram

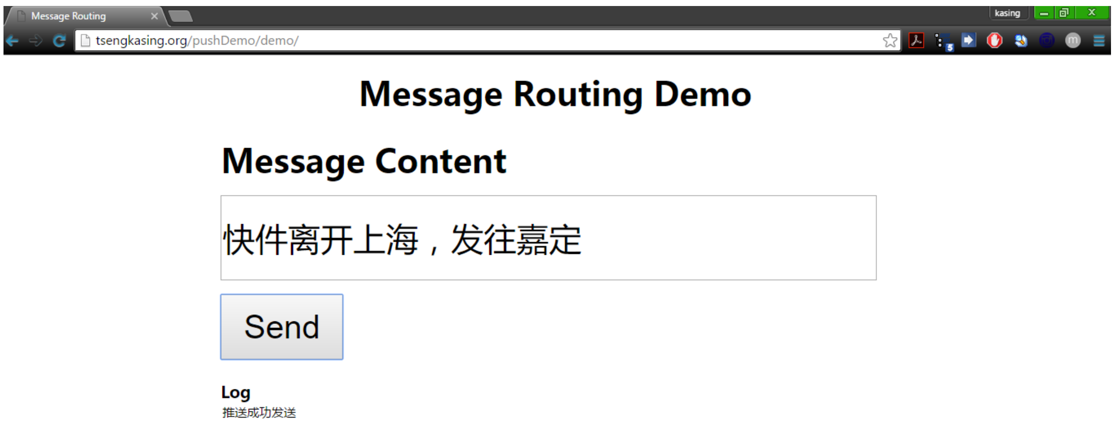
5 Progress of prototyping

5.1 Message Routing Mechanism

Platform

Web + Android

We use a web page as the background controller, administrator can send custom message to a specific channel or even a specific customer with particular ID.



Message Routing Demo

Message Content

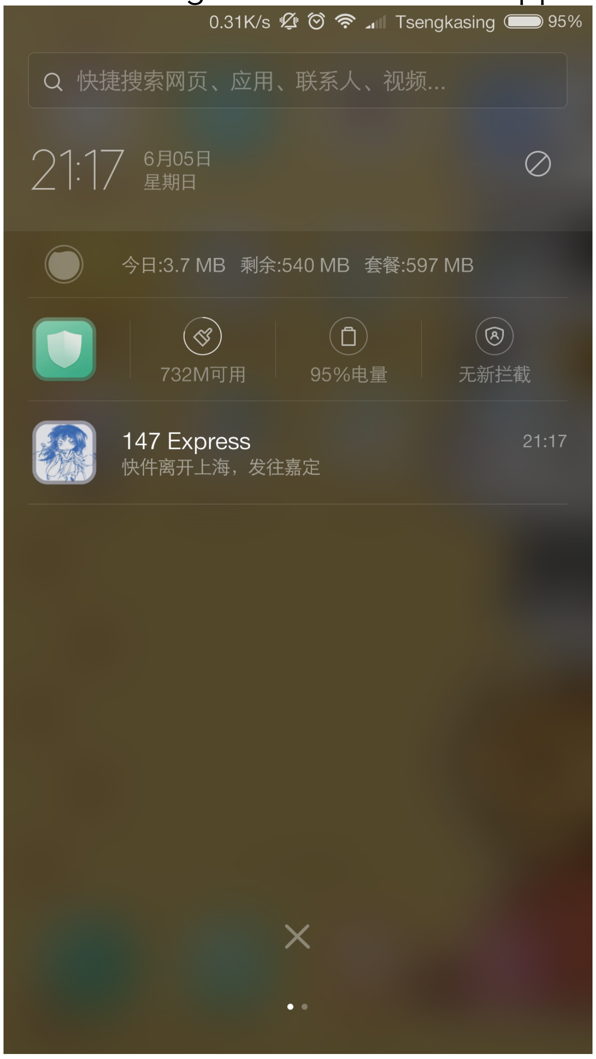
快件离开上海，发往嘉定

Send

Log
推送成功发送

Figure 5.1.1 message routing web page

Customers will receive the message in their Android App on cell phones.


0.31K/s Tsengkasing 95%
快捷搜索网页、应用、联系人、视频...
21:17 6月05日 星期日
今日:3.7 MB 剩余:540 MB 套餐:597 MB
732M可用
95%电量
无新拦截

147 Express 21:17
快件离开上海，发往嘉定

Figure 5.1.2 message routing android client

We realize this function using LeanCloud Service.

Process

Android

As I deploy the SDK of LeanCloud on our Android App and run the receiver service when the App is opended, the App can receive the message we send through the web page.

When the administrator clicked the button "Send", browser will run the JavaScript function and send the message invoking the service of LeanCloud.

Html

```
<input id="message" type="text" />
<button onclick="javascript:createNewMessage()">Send</button>
```

JavaScript

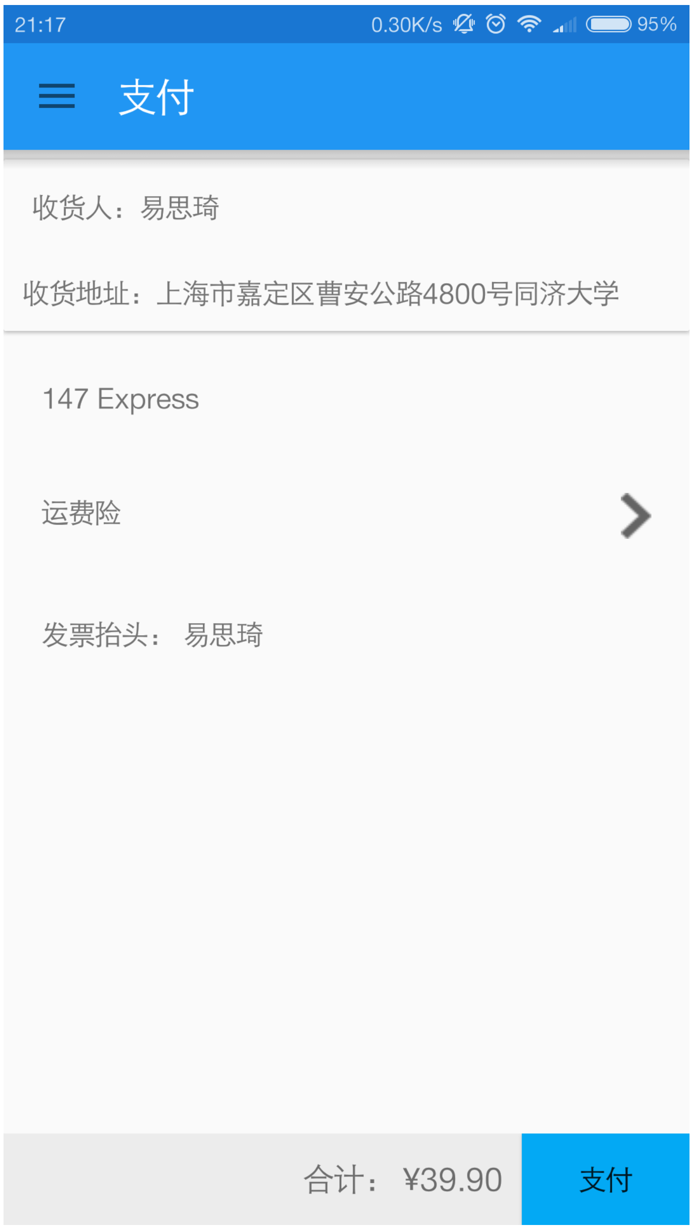
```
function createNewMessage() {
    push = AV.push({
        appId: appId,
        appKey: appKey
    });

    // push
    push.send({
       //set target channel if necessary
        // channels: ['shanghai'],
        data: {alert: document.getElementById("message").value
        }
    }, function(result) {
        if (result) {
            showLog('推送成功发送');
        } else {
            showLog('error');
        }
    });
}
```

5.2 PayByAlipay

Platform and Process

Android

We deploy the Android SDK of alipay on our Android App and it will invoke the service of alipay when the customer press the pay button.



收货人：易思琦

收货地址：上海市嘉定区曹安公路4800号同济大学

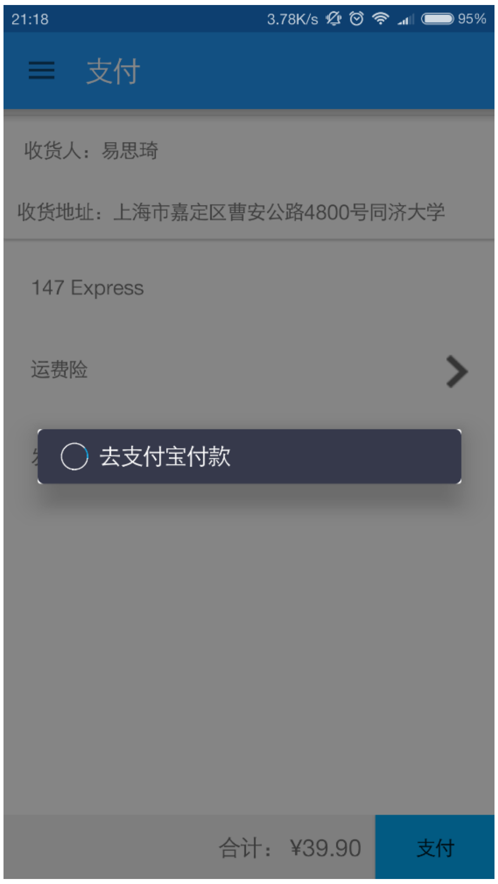
147 Express

运费险

发票抬头： 易思琦

合计： ¥39.90

Figure 5.2.1 alipay android client page 1


21:18 3.78K/s 95%

支付

收货人：易思琦

收货地址：上海市嘉定区曹安公路4800号同济大学

147 Express

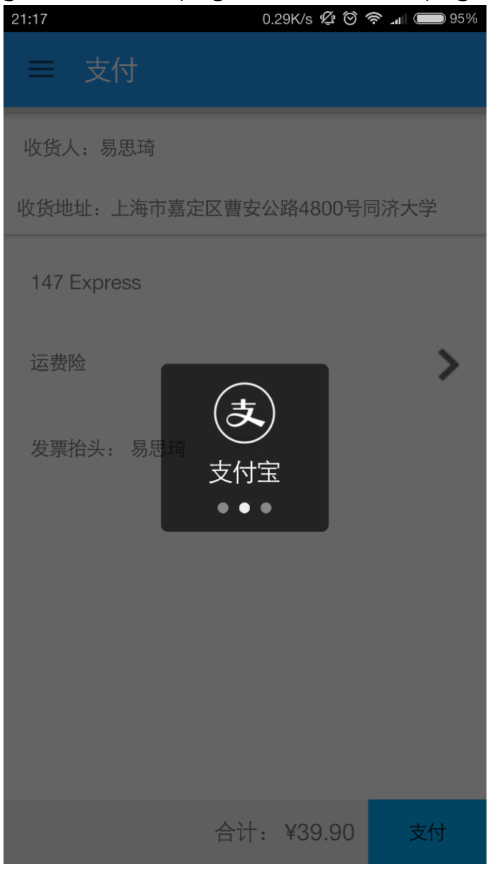
运费险

去支付宝付款

合计：¥39.90

支付

Figure 5.2.2 alipay android client page 2


21:17 0.29K/s 95%

支付

收货人：易思琦

收货地址：上海市嘉定区曹安公路4800号同济大学

147 Express

运费险

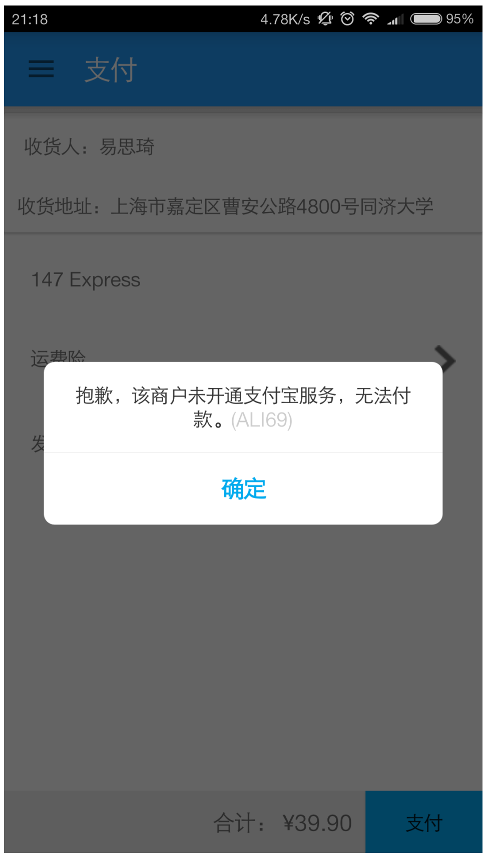
发票抬头：易思琦

支付宝

合计：¥39.90

支付

Figure 5.2.3 alipay android client page 3



收货人：易思琦

收货地址：上海市嘉定区曹安公路4800号同济大学

抱歉，该商户未开通支付宝服务，无法付款。(ALI69)

合计：¥39.90

Figure 5.2.4 alipay android client page 4

It will fail because we don't have the business license to be a signed merchant.

Here is the main code of payment:

```
public void pay() {
    final String payInfo = orderInfo + signInfo;
    // 构造PayTask 对象
    PayTask alipay = new PayTask(getActivity());
    // 调用支付接口，获取支付结果
    String result = alipay.pay(payInfo, true);
}
```

6 Contribution of team members

Yi siqi: Subsystem, Interface Specification, Use Case Realization
Liu lidong: Architecture, Design Mechanism, Use Case Realization
Li Yichao: Subsystem, Interface Specification, Use Case Realization
Zeng Jiaxing: Prototype, Design Mechanism, Use Case Realization

everyone makes equal contribution.