Method of Fare Payment based on RESTful Web Services

Sunhwan Lim[1], Jaeyong Lee[2], and Byungchul Kim[2]

[1] *Internet Service Research Department, ETRI, Daejeon, Korea*
[2] *Dept. of Information Communications Engineering, ChungNam National University, Daejeon, Korea*

shlim@etri.re.kr, {jyl, byckim}@cnu.ac.kr

Abstract

In this paper, the functional architecture for short messaging, payment, and account management RESTful web services was designed that enables IT developers to create applications using telecommunications network elements. Especially, to support a business model that enables operators to offer integrated billing, payment and account management APIs are crucial. In the modeling of short messaging, payment, and account management, we proposed resource definitions and the HTTP verbs applicable for each of these resources. And we measured the TPS of the open service gateway including RESTful web services. Also, using the above model, an example service (i.e. fare payment) consisting of a short messaging service, a payment service, and an account management service was created. Through short messaging, payment, and account management process, the feasibility of the creation of a new service using the proposed architecture and resources was confirmed.

Keywords: *RESTful Open API, Short Messaging, Payment, Account Management*

1. Introduction

Telecommunications networks continually evolve in terms of their form of integrated or converged architecture. From the viewpoint of service, integration between the wire and the wireless services is also a current issue. This type of integration would imply that the end user is provided with seamless broadband multimedia services between wire and wireless networks using the same terminal. The current telecommunications service market is saturated, however. The integration between wire and wireless services provides the opportunity for a new of level services with subscribers using the broadband capability of wired service coupled with the mobility of wireless. The integration between wire and wireless services includes a number of concrete examples that have been developed. Open API (Application Programming Interface) can be easily used to implement or provide integration between wire and wireless services.

Open API is a set of open, standardized interfaces between an application and a telecommunications network [12], [13], [14]. This technology can provide a range of services for the integration of wire and wireless systems independently from network infrastructures, operating systems, or developing languages. In here, use of SOAP (Simple Object Access Protocol) based APIs because of message encoding and decoding, many related stack (e.g. WS security), etc is considered to complex. Alternatively RESTful APIs using HTTP protocol, etc are a light weight. Use of RESTful API would lower the usage barrier for developers from the internet domain, supporting the web 2.0 consumers [6], [7], [8]. OMA (Open Mobile Alliance) defines open APIs (i.e. OMA RESTful network APIs) based on

REST (REpresentational State Transfer) and Parlay defines open APIs (i.e. Parlay X APIs) based on SOAP that enables third party applications to make use of network functionalities [1], [2], [3]. However, the current development status of RESTful open APIs in OMA is ongoing.

In this paper, the functional architecture for short messaging, payment, and account management RESTful web services was designed. The architecture was implemented with Eclipse Galileo version and tested on Apache Geronimo version 2.2. In the modeling of the functional architecture, resource definitions and the HTTP verbs applicable for each of these resources were proposed. And the TPS (Transaction Per Second) of the open service gateway including RESTful web services was measured. Also, using the above model, the functional architecture for an example service (i.e. fare payment) was designed, implemented, and tested.

This paper is organized as follows: Section 2 briefly describes open API. Section 3 details the designed functional architecture for short messaging, payment, and account management RESTful web services, as well as the defined resources for each of these. Section 4 describes the designed functional architecture for an example service (i.e. fare payment). Section 5 describes the implementation of the prototype function and Section 6 is the conclusion.

2. Open API

OMA and Parlay is groups to develop open, technology independent APIs that enable the development of applications capable of operating across converged networks. In these groups, the selection of web services should be driven by commercial utility. The goal is to define a set of powerful simple, highly abstracted, telecommunications capabilities that developers in the IT community can both quickly comprehend and use to generate new, innovative applications. In this section, the Parlay X API (i.e. SOAP based) and OMA RESTful network API is briefly described. A more detailed description of the Parlay X API and OMA RESTful network API is available in the literature [1], [2], [3], [4], [5], [9], [10], [11].

2.1. Parlay X API (SOAP based)

Parlay X APIs should be abstracted from the set of telecommunications capabilities exposed by Parlay/OSA (Open Service Access) APIs (i.e. CORBA based), but may also expose related capabilities that are not currently supported in Parlay/OSA APIs. Parlay/OSA APIs are designed to enable creation of both telephony applications and “telecom enabled” IT applications.

2.2. OMA RESTful Network API

OMA RESTful network APIs define the HTTP protocol binding based on the similar API in Parlay X APIs, using the REST architectural style [6], [7], [8]. These provide resource definitions, the HTTP verbs applicable for each of these resources, and the element data structures, as well as support material including flow diagrams and examples using the various supported message body formats (e.g. XML, JSON, etc) [4].

3. Designed Architecture and Resources for Short Messaging, Payment, and Account Management RESTful Web Services

3.1. High Level Functional Architecture

The high level functional modules of short messaging, payment, and account management RESTful web services are illustrated in Fig. 1. This architecture consists of a web service module, a SCF (Service Capability Feature) module, and an operation and management module. In here, the main reason of the separation between web service module and SCF module is the effective support of services including state information like TPC (Third Party Call), Presence, etc. Web service module only publishes API and SCF module implements service logic of both including (i.e. stateful) and not including state information (i.e. stateless). Alternatively we can only use web service module. However, for the process of services including state information, we may implement service logic using DB including all state information or using request message including all state information parameter. This results in low performance of system. The main functions of each element are described below.

- The web service module receives a request from the AS (Application Server) and forwards the request to the SCF module. This module interacts with the AS using what is known as REST and with the SCF via RMI (Remote Method Invocation).

- The SCF module is the service logic that provides the functionalities for the short messaging, payment, and account management. This module receives a request from the web service module and forwards the request to a SMS-C (Short Message Service - Center) or charging server.

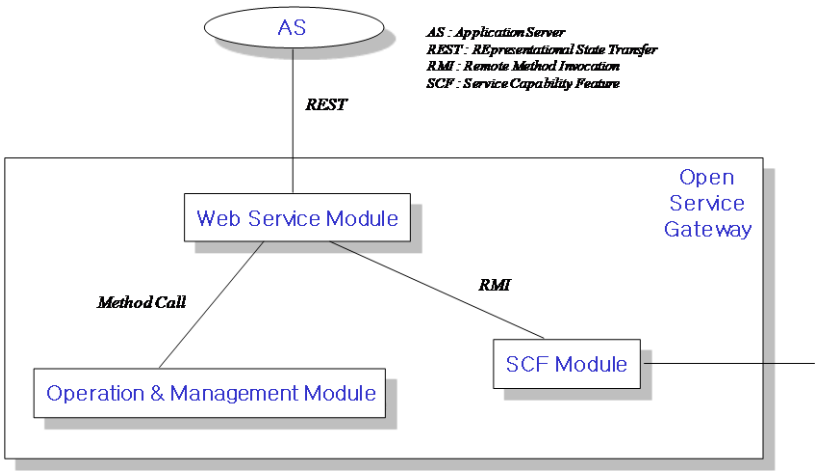

Figure 1. High level functional architecture of short messaging, payment, and account management RESTful web services

- The operation and management module consists of the access control functionality and the log functionality. If any AS requires the use of the short messaging or payment or account management RESTful web service, it first sends a security token that includes the service user's name and password. The access control functionality is the validation of the security token, and the log functionality is a log creation of the controls related to the short messaging or payment or account management.

3.2. Resources for Short Messaging RESTful Web Services

Currently, in order to programmatically send and receive SMS it is necessary to write applications using specific protocols to access SMS functions provided by network elements (e.g. SMS-C). This approach requires a high degree of network expertise. Alternatively it is possible to use open API approach based on web service, invoking standard interfaces to gain access to SMS capabilities. And then, we need light weight RESTful web service.

For the support of short messaging above, in the beginning, we can define SMS message request resource (1), (2) in the resource table below.

In case of HTTP POST method of SMS message requests (1), the resource (http://{server root}/{api version}/shortmessaging/requests) is used to create message request. And in case of HTTP GET method of individual SMS message request (2), the resource (http://{server root}/{api version}/shortmessaging/requests/{requestId}) is used to retrieve an message request including the message delivery status.

Additionally, we need notification subscription resource of SMS message delivery status (3), (4), (5) in the resource table below. Notification subscription resource could give information about SMS message delivery status to the application when a SMS is delivered to a terminal or if delivery was impossible. And also it could be achieved to enhance quality of applications and to allow subscribers to confirm information about SMS message delivery status in real time. The delivery result of the destination address can be one of the following values; Successful delivery to terminal, Successful delivery to network, Unsuccessful delivery, Delivery status unknown, Message is still queued for delivery, and Unable to provide delivery status notification.

In case of HTTP POST method of notification subscriptions of SMS message delivery status (3), the resource (http://{server root}/{api version}/shortmessaging/subscriptions) is used to create a new notification subscription for the particular client. And in case of HTTP PUT method of individual notification subscription of SMS message delivery status (4), the resource (http://{server root}/{api version}/shortmessaging/subscriptions/{subscriptionId}) is used to update an individual subscription for the particular client.

The following table gives a detailed overview of the resources defined for the short messaging RESTful web service. In here, server root is server base URL : hostname+port+base path . For example, http://example.com:8080/exampleAPI.

Table 1. Resources Summary for the Short Messaging RESTful Web Services

Resource	**URL** **Base URL: http://{server root}/{api version}/shortmessaging**	**HTTP verbs**			
		GET	**PUT**	**POST**	**DELETE**
SMS message requests **(1)**	/requests	return all message requests	no	**create new messages request** (requestId assigned)	no
Individual SMS message request **(2)**	/requests/{requestId}	return one message request	no	no	no
SMS message delivery status	/subscriptions	return all subscriptions	no	**create new subscription** (subscriptionId assigned)	no

notification subscriptions **(3)**					
Individual SMS message delivery status notification subscription **(4)**	/subscriptions/{subscriptionId}	return one subscription	**update subscription**	no	**delete one subscription**
Client notification about SMS message delivery status **(5)**	<specified by the client when request is submitted>	no	no	notify client about delivery status	no

3.3. Resources for Payment RESTful Web Services

A vast amount of content, both information and entertainment, will be made available to subscribers. To support a business model that enables operators to offer integrated billing, a payment API is crucial. Open and interoperable "payment APIs" is the key to market growth and investment protection. The payment RESTful web service supports payments for any content in an open, web-like environment. It supports charging of currency amount and volume.

For the support of payment above using RESTful web service, in the beginning, we can define amount and volume charging resource (1), (5) in the resource table below.

In case of HTTP POST method of amount charging (1), the resource (http://{server root}/{api version}/payment/amount /charging) is used to create charging by currency amount.

And also we need split charging resource (2), (6) in the resource table below. If the account of one user is not possible to make payments (i.e. Account balance is low) and the sum of multiple accounts of several users is possible, just then we need split charging method. Split charging could make payments of multiple accounts simultaneously. And also it could be achieved to enhance quality of applications.

In case of HTTP POST method of amount split charging (2), the resource (http://{server root}/{api version}/payment/amount/charging/split) is used to create split charging by currency amount.

Additionally, we need payment reservation resource (3), (4), (7), (8) in the resource table below. This resource results in reserving an amount of an account and charging to a reservation to ensure that the subscriber can fulfill his payment obligations in case of a multimedia service (e.g. a stream of a soccer match).

In case of HTTP POST method of amount reservations (3), the resource (http://{server root}/{api version}/payment/amount/reservations) is used to create the reservation for an account by currency amount. And in case of HTTP POST method of individual amount reservation (4), the resource (http://{server root}/{api version}/payment/amount/reservations/{reservationId}) is used to create charging to a reservation by currency amount.

The following table gives a detailed overview of the resources defined for the payment RESTful web service. In here, server root is server base URL : hostname+port+base path . For example, http://example.com:8080/exampleAPI.

Table 2. Resources Summary for the Payment RESTful Web Services

Resource	URL Base URL: http://{server root}/{api version}/payment	HTTP verbs			
		GET	PUT	POST	DELETE
Amount charging **(1)**	/amount/charging	return transaction by amount	no	**create charging or refunding by amount**	no
Amount split charging **(2)**	/amount/charging/split	return transaction by amount split	no	**create split charging by amount**	no
Amount reservations **(3)**	/amount/reservations	return all reservations	no	**create the reservation by amount** (reservationId assigned)	no
Individual amount reservation **(4)**	/amount/reservations/{reservationId}	return one reservation	**increase or decrease the reservation by amount**	**create charging of the reservation by amount**	**delete one reservation and return funds left in the reservation**
Volume charging **(5)**	/volume/charging	return transaction by volume	no	**create charging or refunding by volume**	no
Volume split charging **(6)**	/volume/charging/split	return transaction by volume split	no	**create split charging by volume**	no
Volume reservations **(7)**	/volume/reservations	return all reservations	no	**create the reservation by volume** (reservationId assigned)	no
Individual volume reservation **(8)**	/volume/reservations/{reservationId}	return one reservation	**increase or decrease the reservation by volume**	**create charging of the reservation by volume**	**delete one reservation and return funds left in the reservation**

3.4. Resources for Account Management RESTful Web Services

Subscribers have credits with their service providers. The consumption of services will lead to reduction of their credit. Therefore, sometimes, subscribers may have to recharge their accounts. This occurs through an application that interfaces with the subscriber either directly or indirectly.

For the support of account management above using RESTful web service, in the beginning, we can define account balance and history resource (1), (2) in the resource table below.

In case of HTTP PUT method of account balance (1), the resource (http://{server root}/{api version}/account/balance) is used to update the account balance. And in case of HTTP GET method of account history (2), the resource (http://{server root}/{api version}/account/history) is used to retrieve the transaction history of the account.

Additionally, we need notification subscription resource of account balance change (3), (4), (5) in the resource table below. Notification subscription resource could give information

about the account changed by some applications (e.g. Multimedia Service, WAP/WEB pages, etc) to other applications (e.g. SMS, MMS, etc) after charge, recharge, and accountLow (i.e. account balance is below the balance threshold). And also it could be achieved to enhance quality of applications and to allow subscribers to confirm information about charges, recharges and accountLow in real time.

In case of HTTP POST method of notification subscriptions of account balance change (3), the resource (http://{server root}/{api version}/account/subscriptions) is used to create a new notification subscription for the particular client. And in case of HTTP PUT method of individual notification subscription of account balance change (4), the resource (http://{server root}/{api version}/account/subscriptions/{subscriptionId}) is used to update an individual subscription for the particular client.

The following table gives a detailed overview of the resources defined for the account management RESTful web service. In here, server root is server base URL : hostname+port+base path . For example, http://example.com:8080/exampleAPI.

Table 3. Resources Summary for the Account Management RESTful Web Services

Resource	**URL** **Base URL: http://{server root}/{api version}/account**	**HTTP verbs**			
		GET	**PUT**	**POST**	**DELETE**
Account balance **(1)**	/balance	**return the account balance and the expiration date**	**update the account balance**	no	no
Account history **(2)**	/history	**return the transaction history of the account**	no	no	no
Account balance change notification subscriptions **(3)**	/subscriptions	return all subscriptions	no	**create new subscription** (subscriptionId assigned)	no
Individual account balance change notification subscription **(4)**	/subscriptions/{subscriptionId}	return one subscription	**update subscription**	no	**delete one subscription**
Client notification about account balance change **(5)**	<specified by the client when request is submitted>	no	no	notify client about account balance change	no

4. Designed Architecture for Example Service (Fare Payment)

4.1. Functional Architecture for Fare Payment

Functional blocks of open service application server and gateway for fare payment using open API are illustrated in Fig. 2. Open service application server provides service user with payment transaction history retrieving functionality. From here, we can know that open

service application server provides service user with payment transaction history retrieving UI (i.e. account management User Interface). And it also stores the customer data for service user subscription management. If service user requests the payment transaction history, account management logic in open service application server requests the act for payment transaction history of open service gateway using account management API. Open service gateway account management functionality performs the request from account management logic. And then the result forwards the payment transaction history in charging server to account management logic in open service application server. Payment transaction history forwarded to account management logic in open service application server is provided with service user through account management UI.

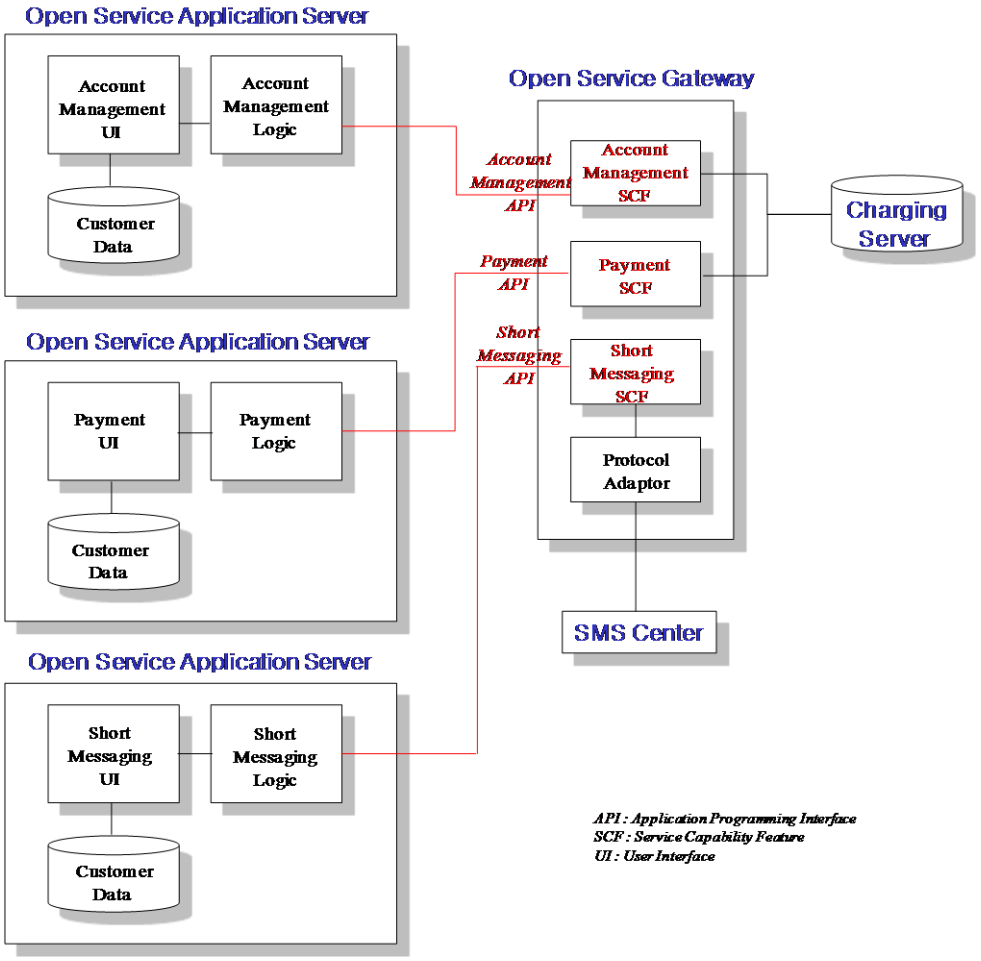

Figure 2. Functional Architecture for Fare Payment

From Fig. 2, we can know that open service application server provides service user with payment UI. And it also stores the customer data for service user subscription management. If service user requests the fare payment, payment logic in open service application server requests the act for payment of open service gateway using payment API. Open service gateway payment functionality performs the request from payment logic. And then the result is applicable to charging server. The charging server stores payment transaction data and log.

From Fig. 2, we can know that open service application server provides service user with short messaging UI. And it also stores the customer data for service user subscription management. If service user requests the short messaging, short messaging logic in open service application server requests the act for short messaging of open service gateway using short messaging API. Open service gateway short messaging functionality performs the request from short messaging logic. And then the result sends the short message to mobile phone through SMS center.

4.2. Scenario Flow for Fare Payment

Payment and SMS scenario flow for fare payment service is illustrated in Fig. 3. If fare payment is requested from service user in init state, it receives service user mobile phone number and also receives service user account number. Using input data, it forwards fare payment request to open service gateway in telecommunication network. If the result is YES, it sends the result using SMS to mobile phone. If a balance associated with the account is below the balance threshold, it sends account balance using SMS to mobile phone. If the account is unregistered, it sends unregistered account message using SMS to mobile phone.

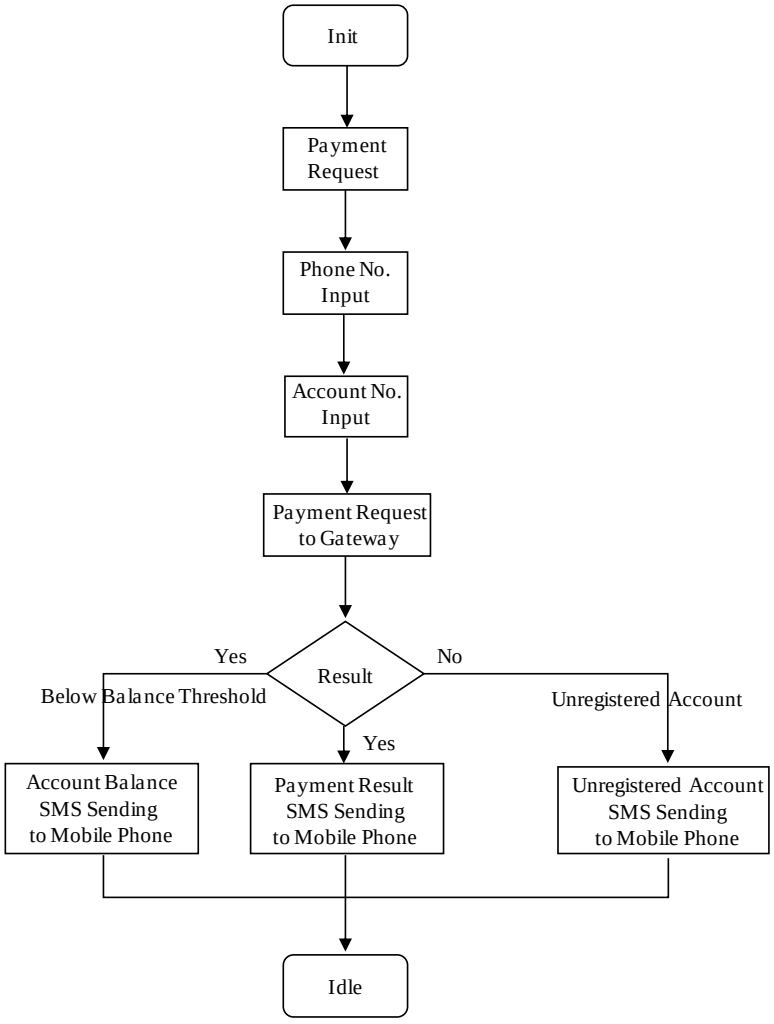

Figure 3. Flow of Payment and SMS for Fare Payment

Account management scenario flow for fare payment service is illustrated in Fig. 4. If fare payment transaction history is requested from service user in init state, it receives service user account number and also receives password. Using input data, it forwards payment transaction history request to open service gateway in telecommunication network. If the result is YES, it displays the payment transaction history. If service user's password is wrong, it displays password error message. If the account is unregistered, it displays unregistered account message.

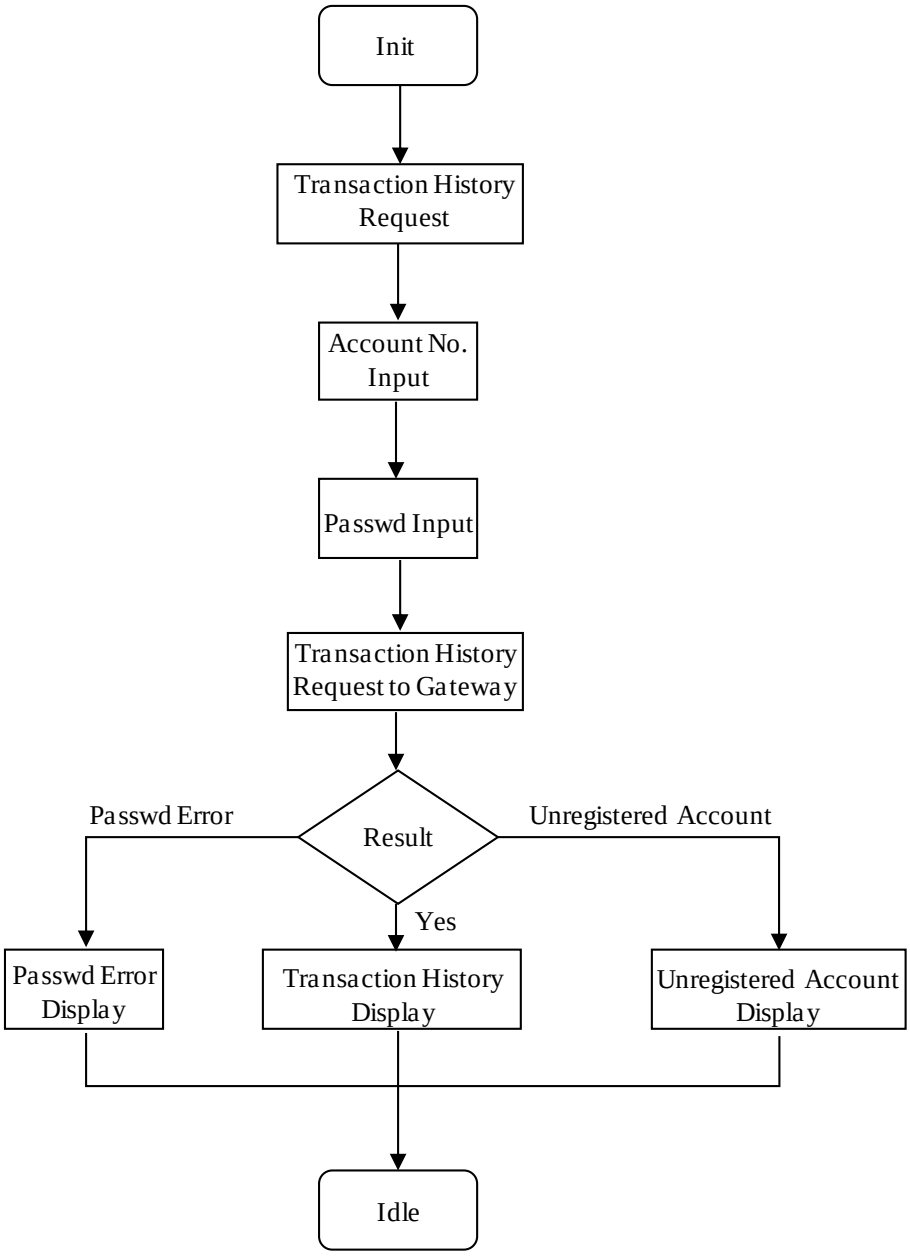

Figure 4. Flow of Account Management for Fare Payment

5. Implementation of the Prototype Function

5.1. Environments and Testing

Short messaging, payment, and account management RESTful web services were implemented using Eclipse Galileo version and tested on Apache Geronimo version 2.2. These RESTful web services consisted of a web service module and a SCF module. The web service module interacts with the SCF module using RMI, which implies that a security policy between these modules is needed. However, a security policy is in fact unnecessary, as the two modules are parts of the same system. And also these RESTful web services interact with a Mysql DB to record transaction history information using JDBC (Java Database Connectivity), with SMS-C for sending a SMS message to a terminal, and with the charging server for the charging or billing information.

For the TPS measurement of the open service gateway including RESTful web services, the above three and additional four web services were tested using IBM Rational Performance Tester version 8.2.

The following table gives the TPS for the selected API in the web services of the open service gateway.

Table 4. TPS for open service gateway

Service Component	API	TPS (Average : 249.62)
SMS	sendSMS	252
Payment	chargeAmount	254.5
Account Management	getBalance	253
Presence	getBuddyList	256.2
Directory	getContactList	239
Third Party Call	makeCall	208
Mail	sendMail	284.7

An example service (i.e. fare payment) was used that consisted of a short messaging, a payment service, and an account management service. The created services were implemented and tested on Microsoft Visual Studio 2010 using the C# language.

6. Conclusion

The current telecommunications market is saturated. Regarding new market growth, a range of new intelligent services is on the horizon. Potential subscribers must be introduced to these services, but it is currently not feasible to bring third party service providers and developers into the vertical architecture of current telecommunications networks. Thus, open, technology independent APIs that enable the development of applications that operate across converged networks are necessary.

In this paper, the functional architecture for short messaging, payment, and account management RESTful web services was designed that enables IT developers to create applications using telecommunications network elements. The architecture was implemented with Eclipse Galileo version and tested on Apache Geronimo version 2.2. In the modeling of the functional architecture, resource definitions and the HTTP verbs applicable for each of these resources were proposed. And the TPS of the open service gateway including RESTful web services was measured. Also, using the above model, the functional architecture for an

example service (i.e. fare payment) was designed, implemented, and tested. Through the short messaging, the payment, and the account management process, the feasibility of the creation of a new service using the proposed architecture and resources was confirmed.

Acknowledgment

"This research was supported by the KCC (Korea Communications Commission), Korea, under the R&D program supervised by the KCA (Korea Communications Agency)" (KCA-2011- (09913-05001))

This paper is a revised and expanded version of a paper [Open API and System of Short Messaging, Payment, Account Management based on RESTful Web Services] presented at [ISA & ACN 2011, Aug. 15 – 17 2011, Brno Czech]

References

[1] 3GPP, Third Generation Partnership Project, http://www.3gpp.org/
[2] OMA (Open Mobile Alliance), http://www. openmobilealliance.org/
[3] ETSI, European Telecommunications Standards Institute, http://www.etsi.org/
[4] W3C, World Wide Web Consortium, http://www.w3c.org/
[5] IETF, Internet Engineering Task Force, http://www.ietf.org/
[6] Roy Fielding, "Architectural Styles and the Design of Network based Software Architecture", Dissertation of Doctor of Philosophy in Information and Computer Science, University of California, IRVINE (2000)
[7] Leonard Richardson and Sam Ruby, RESTful Web Services, O'Reilly Media, (2007-5)
[8] Cesare Pautasso, "REST vs. SOAP: Making the right architectural decision", 1st International SOA Symposium (2008-7)
[9] Parlay X Working Group, Parlay X Web Services White Paper v1.0 (2002)
[10] Web Services Working Group, Parlay Web Services WSDL Style Guide (2002)
[11] Parlay X Working Group, Parlay X Web Services Specification v1.0 (2003)
[12] ArdJan Moerdijk and Lucas Klostermann, Ericsson Eurolab Netherlands, "Opening the Networks with Parlay/OSA: Standards and Aspects behind the APIs", IEEE Network (2003)
[13] WeiWu, Hua Zou, Fangchun Yang, "Design OSA/Parlay Application Frameworks Using a Pattern Language", Proceedings of ICCT (2003)
[14] J.W. Hellenthal, F.J.M. Panken, M. Wegdam, "Validation of the Parlay API through Prototyping", IEEE Intelligent Network Workshop (2001)
[15] Karsten Luttge, "E-Charging API: Outsource Charging to a Payment Service Provider", IEEE Intelligent Network Workshop (2001)

Authors

Sunhwan Lim is currently a Senior Member of Engineering Staff at the Department of Internet Service Research of Electronics and Telecommunications Research Institute, Korea, since 1999. His research interests include open API, web services, service and mobile platform, smart work.

Jaeyong LEE received the B.S. degree in Electronics engineering from Seoul National University and M.S. and Ph.D degrees in electronic engineering from Korea Advanced Institute of Science and Technology (KAIST), Korea, in 1988, 1990 and 1995. He is currently a professor at the Department of Information and Communication Engineering of Chungnam National University, Korea since 1995. Also, from 1990 to 1995, he worked as a research engineer at the Digicom Institute of Information and Communications. His research interests include Internet protocols, traffic control, performance analysis and mobile internet.

Byungchul KIM received the B.S. degree in Electronics engineering from Seoul National University and M.S. and Ph.D degrees in electronic engineering from Korea Advanced Institute of Science and Technology (KAIST), Korea, in 1988, 1990 and 1996. He is currently a professor at the Department of Information and Communication Engineering of Chungnam National University, Korea since 1999. Also, from 1993 to 1999, he worked as a Research Engineer at the Samsung Electronics. His research interests include computer networks, wireless internet, sensor networks and mobile communications.