International Journal of Internet, Broadcasting and Communication Vol.11 No.1 8-16 (2019)
http://dx.doi.org/10.7236/IJIBC.2019.11.1.8

IJIBC 19-1-2

Designing a Comet-based Open API for Establishing RCS Chat Session

Dongcheul Lee[1] and Byungjoo Park[2*]

[1,2]*Department of Multimedia Engineering, Hannam University, Korea*
[1]*jackdclee@hnu.kr,* [2*]*bjpark@hnu.kr*

Abstract

As smartphone users grow, mobile operators are trying to standardize and commercialize Rich Communication Suite (RCS), which is a next-generation messaging service, so that it can replace legacy messaging services. However, it is not enough to spread RCS widely to the users only by publishing an RCS app. To increase the use of RCS, a web-based open API for common RCS capabilities is needed. By using the API, Internet-based developers can create applications that make use of the RCS capabilities with less effort and time. This paper proposes a lightweight Comet-based open API to allow mobile operators to expose useful information and capabilities to application developers. The system architecture of the open API framework and call flow between relevant nodes are defined. In addition, examples of protocol translations on the framework are provided.

Keywords: *RCS, open API, Comet, software framework*

1. Introduction

As smartphones became popular, mobile phone users have changed their ways of texting with other people. Nowadays, they are using Over-The-Top (OTT) services like Kakao Talk or Line instead of Short Message Service (SMS) or Multimedia Message Service (MMS) [1]. This means that telecommunication companies are losing their revenue streams because SMS and MMS were one of their main sources of profit. For developing new sources of revenue they have proposed the next generation messaging service with Groupe Speciale Mobile Association (GSMA) [2]. This service is called Rich Communication Suite (RCS) and they are trying to standardize the service between all telecommunication companies over the world and are trying to embed it as a default messaging app on smartphones [3].

However, these strategies have limitations for spreading the use of RCS. Mobile phone users need smartphones and RCS apps if they want to use RCS. To spread the use of RCS, users should be able to use it on various apps on multi-platforms. Therefore, telecommunication companies need to open their RCS infra-systems to other 3rd party RCS-based app developers.

Manuscript Received: Dec. 15, 2018 / Revised: Dec. 20, 2018 / Accepted: Dec. 27, 2018
Corresponding Author: bjpark@hnu.kr
Tel: +82-42-629-8489, Fax: +82-42-629-8489
Department of Multimedia Engineering, Hannam University, Korea

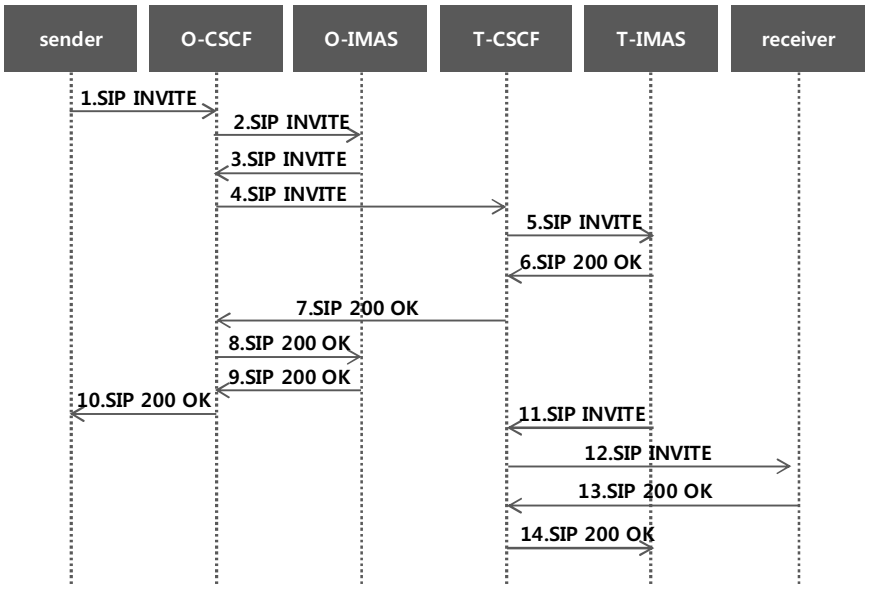

Figure 1. Call flow for RCS-e messages

To make RCS infra-systems easily accessible, telecommunication companies should provide the web-based open Application Program Interface (API) to 3rd party developers [4]. Existing RCS clients are using complex Internet Protocol Multimedia Subsystems (IMS) protocols such as Session Initiation Protocol (SIP), Message Session Relay Protocol (MSRP), and Extensible Markup Language Configuration Access Protocol (XCAP). However, it is very hard for 3rd party developers to implement these IMS protocol stacks. If they can use lite-weight web-based RCS open API on their apps, they can develop RCS-based apps without the knowledge of IMS protocols [5, 6, 7]. This paper designs a web-based RCS open API framework. By using this framework, 3rd party RCS developers can develop their apps fast and efficiently.

The next section introduces related works on an RCS message flow when sending a message. In addition, the Comet model is introduced which enables a web server to send data to a client in real-time. In section 3, we propose the architecture of the open API framework and design the message flow between the framework and existing IMS nodes. How protocol translation is performed in the framework is also discussed. The paper concludes with a short discussion.

2. Related Works

2.1 RCS-e

Joyn is a GSMA-launched market brand for RCS 1.2.2 specification, which is also called RCS-e [8]. The RCS-e specification uses the call flow in Figure 1 when a sender initially sends a message to a receiver. Detail description of the flow is as follows:

(1) A sender types a text to send using the RCS app. When the sender presses a send button, the RCS app sends a SIP INVITE message to an Originating Call Session Control Function (O-CSCF). The sender had sent a SIP REGISTER message to the O-CSCF to register its information before it sends the SIP INVITE message.

(2) The O-CSCF sends a SIP INVITE message to an Originating Instant Messaging Application Server (O-IMAS) using the initial Filter Criteria (iFC).

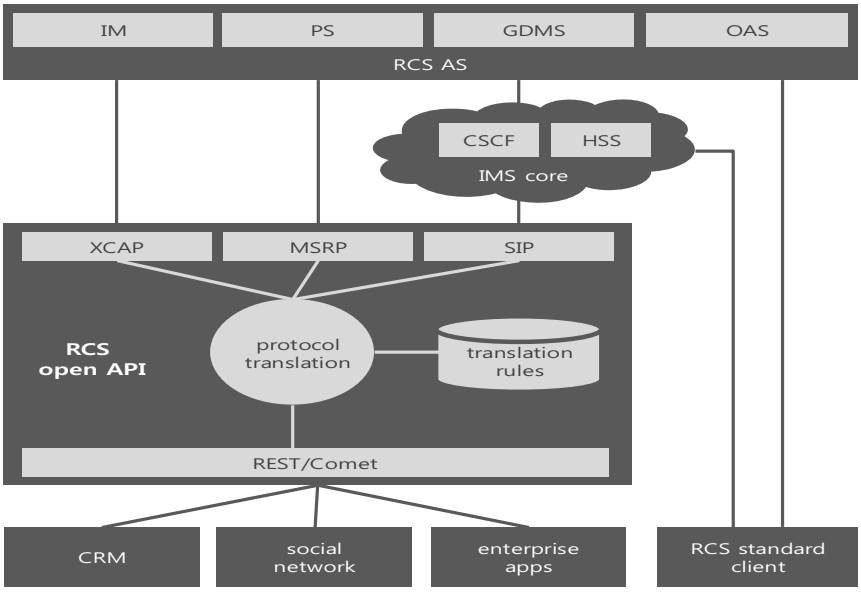

Figure 2. RCS open API system architecture

(3) The O-IMAS sends a SIP INVITE message to the O-CSCF to send the text to a CSCF that the receiver has registered.

(4) The O-CSCF sends a SIP INVITE message to a Terminating Call Session Control Function (T-CSCF) using its routing table.

(5) The T-CSCF sends a SIP INVITE message to a Terminating Instant Messaging Application Server (T-IMAS).

(6-10) The T-IMAS send a SIP 200 OK message for acknowledging the SIP INVITE message along the path that the SIP INVITE message came from.

(11-14) The T-IMAS sends a SIP INVITE message to the T-CSCF where the receiver registered. Then the T-CSCF sends a SIP INVITE message to the receiver's app. The app sends a SIP 200 OK message for acknowledging the SIP INVITE message along the path that the SIP INVITE message came from.

2.2 Comet

Comet is a web app model that can allow a web server to push data to a web client without the client explicitly requesting it [9]. In the original model of the web, the client should send a request to get a complete web page at a time. By using Comet, the client can receive data from the server in real-time. This model is also called Asynchronous JavaScript and eXtensible Markup Language (Ajax) Push, Reverse Ajax, or Two-way-web. There are various ways to implement the model, but major ways can be divided into the streaming and the long polling.

A web app using the streaming opens a single persistent connection with the web server for all Comet events. The app sends a hidden iframe Hypertext Transfer Protocol (HTML) element as a chunked block,

which indicates it as infinitely long. When there are events to send to the app, the iframe is filled with script tags, containing JavaScript to be executed in the app. As the app receives it, the JavaScript in the iframe is executed. A web app using the long polling makes an asynchronous request of the server waiting for data to be available before responding. When there are events to send to the app, the server responds with

XML, or JavaScript Object Notation (JSON) encoded data or JavaScript that can be executed in the app. After processing the response, the app creates another request to await the next event. Thus, the app always keeps a request that can be answered as each event occurs.

3. Proposed Framework

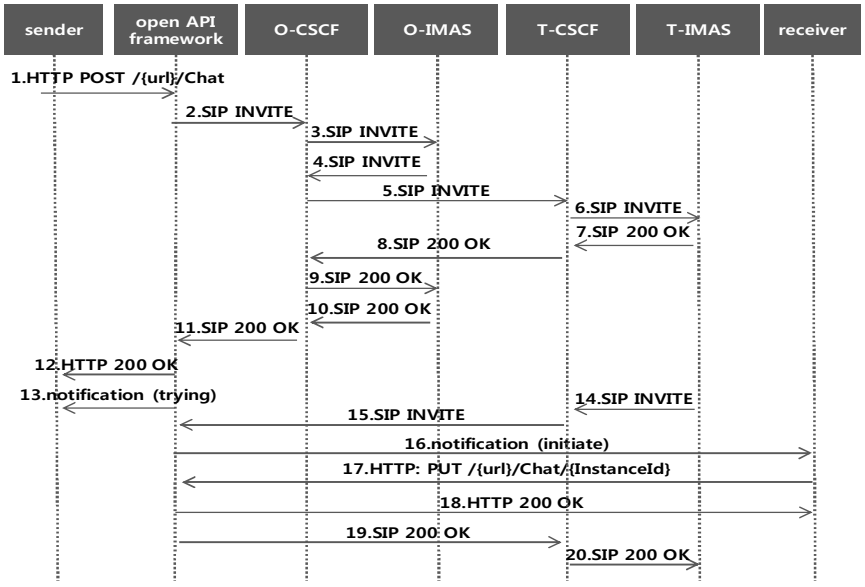

Figure 3. Call flow for chat session establishment using the RCS open API framework

The web-based RCS open API framework enables 3rd party app developers to make apps that use RCS infra-systems. We can consider various web app models to provide RCS as a web-based interface. In this paper, we chose Comet and Representational State Transfer (REST) as a baseline web app model. REST is a software architecture style that defines information as web resources. The resources can be accessed via Uniform Resource Locator (URL), which leads to simplicity and extensibility and it is suitable for RCS which needs fast and stable services [10]. In addition, mobile apps can use it easily since they only need to make HTTP connections. However, REST does not provide real-time data transfer so therefore we also need to adopt Comet to send an RCS message in real-time.

Figure 2 shows the system architecture of the proposed RCS open API framework. There are various existing Application Servers (AS) to provide RCS which include: Instant Messaging (IM), Presence Server (PS), Generic Data Manipulation Server (GDMS), and OPTIONS Application Server (OAS) etc. IM manages instant messages between the users, and PS manages presence information of the users. GDMS manages users' profiles and OAS manages SIP OPTIONS messages. These ASs are using SIP, MSRP, and XCAP to provide RCS. Therefore, the RCS open API framework should implement these protocol stacks to interoperate with the ASs. SIP, MSRP, and XCAP messages are converted to REST-and-Comet-based HTTP messages according to the protocol translation rules on the RCS open API framework. These converted HTTP messages can be sent to a Customer Relationship Management (CRM), a social network service, or a business app. Meanwhile, existing RCS apps send a SIP message to the AS through the IMS core network and they can send an MSRP and XCAP message to the AS directly. Therefore, using the RCS open API does not affect existing RCS protocols.

Figure 3 shows the call flow for a chat session establishment using the proposed open API framework.

The sender and the receiver in the figure use the RCS open API client. Detail description of the flow is:

Table 1. Example of translating an HTTP POST request to a SIP INVITE request

HTTP POST	SIP INVITE

POST /1.0/RCS/01087654321/Chat HTTP/1.1
HOST: 1.1.1.1:1001
Content-Type: application/json;charset=utf-8
Content-Length: 183
Connection: Keep-Alive
 	
{
"data": {
"message_id": "ASedjkffjksjldf",
"recipient": [
"01012345678"
],
"subject": “Hi, I'm Jack."
},
"oauth_token":
"lkasjdfsSADJFlksajdfSADfalsikjufi34roihjfglasSDF3i"
}
	INVITE sip:01012345678@domain.com SIP/2.0
	To: <sip:01012345678@domain.com>
	Via: SIP/2.0/UDP 1.1.1.2:1002
	CSeq: 1 INVITE
	From: <sip:01087654321@domain.com>
	Contact: <sip:1.1.1.2:1002>
	Subject: Hi, I'm Jack.
	Content-Type: multipart/mixed;boundary=boundary1
	Content-Length: 501
	Accept-Contact: *;+g.oma.sip-im;explicit;require
	Session-Expires: 1800;refresher=uac
	User-Agent: Open API;service="RCS-e"
	--boundary1
	Content-Type: message/cpim
	Content-Length: 316
	From: <sip:01087654321@domain.com>
	To: <sip:01012345678@domain.com>
	NS: imdn <urn:ietf:params:imdn>
	imdn.Message-ID: ASedjkffjksjldf
	DateTime: 2012-01-01T10:10:10
	imdn.Disposition-Notification: positive-delivery, display
	Content-type: text/plain
	Hi, I'm Jack.
	--boundary1
	Content-Type: application/sdp
	(sdp body here)
	--boundary1--

Table 2. Example of translating a SIP INVITE request to a Comet-based HTTP response

SIP INVITE	Comet HTTP response

INVITE sip:01012345678@domain.com SIP/2.0
To: sip:01012345678@domain.com
Via: SIP/2.0/TCP 1.1.1.3:1003,SIP/2.0/TCP 1.1.1.4:1004
Cseq: 1005 INVITE
From: sip:01087654321@domain.com
Contact: <sip:1.1.1.4@1.1.1.4:1004>
Subject: Hi, I'm Jack.
Content-Type: multipart/mixed;boundary=boundary1
Content-Length: 679
Accept-Contact: *;+g.oma.sip-im;explicit;require
--boundary1
Content-Type: message/cpim
Content-Length: 255
From: <sip:01087654321@domain.com>
To: <sip:01012345678@domain.com>
NS: imdn <urn:ietf:params:imdn>
imdn.Message-ID: ASedjkffjksjldf
DateTime: 2012-01-01T10:10:10
imdn.Disposition-Notification: positive-delivery, display
Content-type: text/plain
Hi, I'm Jack.
--boundary1
Content-Type: application/sdp
(sdp body here)
--boundary1--
	{
	"type" : "chat",
	"instance" : "http://1.1.1.1:1001/1.0/RCS/01012345678/Chat/01012345678-001",
	"data" : {
	"status" : "initiate",
	"type" : "text",
	"subject" : “Hi, I'm Jack.",
	"message" : “Hi, I'm Jack.",
	"message_id" : "ASedjkffjksjldf",
	"sender" : "01087654321"
	}
	}

(1) RCS open API client opens a Comet connection with the framework. It chooses a resource using HTTP POST message and sends the text using JSON [11]. It uses the following format when it sends HTTP POST:

POST /{version}/RCS/{RCS ID}/Chat

{version} contains open API's version and {RCS ID} contains sender's RCS ID. A JSON message has data objects which have following keys: a message_id key which uniquely identifies each message, a recipient key which is receiver's ID, a subject key which is a content of the text, and an oauth_token key which is the authentication token issued when the client successfully authenticated before using the open API. The left column of Table 1 shows an HTTP POST message example when a sender (01087654321) sends

"Hi, I'm Jack." to a receiver (01012345678).

(2-3) Sender's HTTP POST message is converted to a SIP INVITE message according to the translation rules on the open API framework. Sender's ID and Receiver's ID are converted to SIP URI format and stored in From header and To header respectively. The value in the message_id field is stored in imdn.Message-ID field. The value in the subject field is stored in Instant Message Disposition Notifications (IMDN) field [12]. The right column of Table 1 shows how to convert an HTTP POST message in the left

column to a SIP INVITE message. The framework sends the converted SIP INVITE message to an O-IMAS

through an O-CSCF in the IMS core network.

(4-6) The O-IMAS sends a SIP INVITE message to a T-IMAS in the same ways with (3-5) in Figure 1.

(7-11) The T-IMAS sends a SIP 200 OK message to the framework in the same ways with (6-10) in Figure 1.

(12) The framework sends an HTTP 200 OK message to the sender as a response to the HTTP POST message.

(13) The framework notifies the sender that the SIP INVITE message has been a trying state using Comet.

(14-15) The T-IMAS sends a SIP INVITE message to the framework in the same ways with (11-12) in Figure 1. The left column of Table 2 shows an example of the message.

(16) The framework converts the SIP INVITE message to a Comet-based HTTP response message. The framework set following values as JSON keys: a Type field is set to "chat" and an Instance field is set to the instance ID generated from the framework. The instance ID is used when the client sends a message to the framework for managing a conversation session. The ID has the following format:

http://{server IP address}/{version}/RCS/{RCS ID}/Chat/{instance-id}

The data object has the following keys: Status, Type, Subject, Message, Message_id, and Sender. The Status field is set to "initiate", meaning that it is the request for initiating a session. The Type field is set to

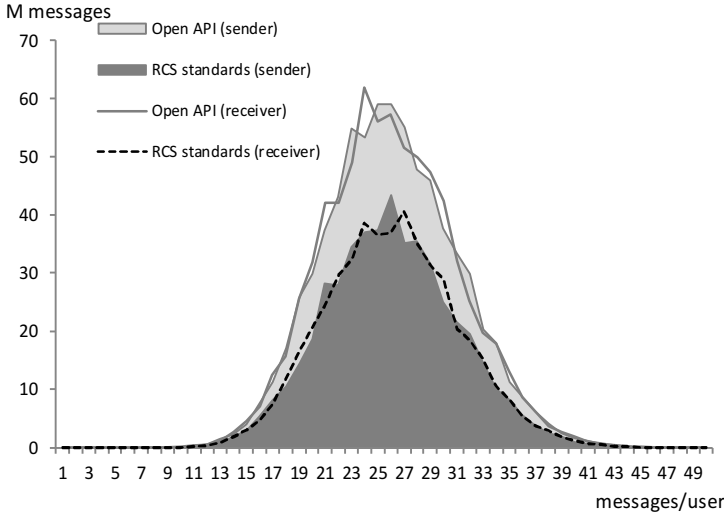

Figure 4. Simulation results using the open API framework and the standard scheme

"text", meaning that the message is a plain text format. The Subject field is set to the title of the message. The Message field is set to the body of the message. The Message_id field is set to the value of Message_id that the sender used. The Sender field is set to the sender's ID. The right column of Table 2 shows an

example that the SIP INVITE message is converted to the JSON format.

(17) The receiver sends an HTTP PUT message to the framework for acknowledging the conversation session request. The message has the following format:

PUT /1.0/RCS/{RCS ID}/Chat/{instance-id} HTTP/1.1

The body of the message contains Oauth_token value so that it can be used for authentication [13].

(18) The framework sends an HTTP 200 OK message to the receiver as a response to the HTTP PUT message.

(19-20) The framework sends a SIP 200 OK message through the T-CSCF in order to notify that the receiver approved opening the conversation session.

4. Performance Evaluation

To evaluate the performance of the proposed framework, we compared the number of messages sent from and received at user's device when the framework used and when it was not used. Since the framework works as a standard RCS client, the message flow between CSCFs and ASs was not changed. We used Figure 1 and Figure 3 to make a simulation. We build a C++ program to emulate the nodes in Figure 2. We assumed that there are 10 million RCS users and those users send X number of RCS text messages per day where X is a Poison random variable with an interval 25. Figure 4 shows the result of the simulation. The senders and the receivers use 3 messages when they initiate a chat session using the framework whereas they use 2 messages when they use standard clients. This means that the open API clients should process 1.5 times more messages than the standard clients.

5. Conclusions

This paper designed the web-based RCS open API for spreading the use of RCS. Sending a text to others in real-time is important in RCS. Therefore, we used the Comet model so that the API framework can send a text to a client in real-time. We defined the system architecture to implement the framework and designed the call flow between the framework and related IMS nodes. In addition, we provided examples for converting IMS protocols to Comet-based HTTP messages or vice versa. The simulation result shows that the open API client should process 1.5 times more messages than the standard client when initiate a chat session. However, existing IMS core nodes process the same number of messages regardless of the clients. This means that telecommunication companies do not have to make an investment on IMS core nodes to provide the open API services. Our future work includes implementing the framework so that the 3rd party RCS app developers can make use of this.

Acknowledgement

This work was supported by Hannam University Research Fund in 2018.

References

[1] B. B. Moshe, A. Dvir, and A. Solomon, "Analysis and optimization of live streaming for over the top video", in *Proc. of the 2011 IEEE Consumer Communications and Networking Conference*, pp. 60-64, 2011. DOI: 10.1109/CCNC.2011.5766553

[2] T.V. Pelt, "Rich Communication Suite 6.0 Advanced Communications Services and Client Specification", in *GSM Association*, pp. 9-18, March 2016.

[3] E. Ersoz, "RCS Device API 1.6 Specification Version 4.0", in *GSM Association*, pp. 5-8, March 2016.

[4] Y. Park, Y. Choi and S.-H. Kim, "Presence-based call reservation service using open API on the Web-service architecture in broadband converged network", in *Proc of the 8th International Conference on Advanced Communication Technology*, Vol. 1, 2006.
DOI: 10.1109/ICACT.2006.205967

[5] J. Rosenberg, et. al., "SIP: Session Initiation Protocol, Request for Comments: 3261", in *IETF*, 2002.

[6] B. Campbell, et al., "The Message Session Relay Protocol, Request for Comments: 4975", in *IETF*, 2007.

[7] J. Rosenberg, "The Extensible Markup Language (XML) Configuration Access Protocol (XCAP), Request for Comments: 4825", in *IETF*, 2007.

[8] T.V Pelt, "RCS-e - Advanced Communications: Services and Client Specification Version 1.2.1", in *GSM Association*, 2011.

[9] H. Ren, R. Pi, J. Chi, X. Wang and L. Lin, "Collaborative Web Page Browse with AJAX and Comet Proxy", in *Proc. of the Third International Conference on Pervasive Computing and Applications*, Vol. 2, pp. 715-717, 2008.
DOI: 10.1109/ICPCA.2008.4783702

[10] X. Shi, "Sharing service semantics using SOAP-based and REST Web services", in *IT Professional*, Vol. 8, Iss. 2, pp. 18-24, 2006.
DOI: 10.1109/MITP.2006.48

[11] Y. Jun, L. Zhishu and M. Yanyan, "JSON Based Decentralized SSO Security Architecture in E-Commerce", in *Proc. of the International Symposium on Electronic Commerce and Security*, pp. 471-475, 2008.
DOI: 10.1109/ISECS.2008.171

[12] E. Burger and H. Khartabil, "Instant Message Disposition Notification, Request for Comments: 5438", in *IETF*, 2009.

[13] R. E. Navas and L. Toutain, "LATe: A Lightweight Authenticated Time Synchronization Protocol for IoT", in *Proc of 2018 Global Internet of Things Summit (GIoTS)*, pp. 1-6, 2018.
DOI: 10.1109/GIOTS.2018.8534565