This paper was published in IEEE INTERNET COMPUTING, JULY/AUGUST 2012. This is an author copy for personal record only.

SOAP-Based Web Services vs. RESTful Web Services for Multimedia Conferencing Applications: A Case Study

Fatna Belqasmi, Jagdeep Singh, Suhib Bani melhem, Roch H. Glitho

Abstract— **RESTful web services are now emerging as an alternative that may be more suitable than SOAP-based counterparts in some cases. In this paper, we contrast these two web programmatic interfaces for the development of multimedia conferencing applications, an important category of web applications. A RESTful web service that offers the same functionality as the standard Parlay-X multimedia SOAP-based web service is proposed. The two interfaces are prototyped in the same environment, and the same application is developed using both interfaces. The performance of each are compared and lessons learned are discussed.**

Key words: **SOAP-based web services, RESTful web services, Multimedia conferencing, Parlay-X web services**

I. INTRODUCTION

Multimedia conferencing applications are an important category of web applications. Some examples are audio/video conferencing, multiparty on-line games and distance learning. Like all web applications, they can be developed using either Simple Object Access Protocol (SOAP)-based web services or RESTful web services as the programmatic interface. The SOAP-based web service architecture [1] comprises three entities: service provider, service registry, and service requestor. The service provider creates a SOAP-based web service and publishes the service description in the service registry. The service requestor finds the service by querying the service registry. It retrieves the service description, uses it to bind to the service implementation, and then starts interacting with it. The service registry enables the on-line discovery of services. The communications (operations) among the three web service entities rely on XML and use SOAP. SOAP messages are commonly exchanged over HTTP, even though other bindings are possible. The service descriptions are published using the Web Services Description Language (WSDL). WSDL provides information on how to use a web service, including a description of the service methods and binding information. Universal Description, Discovery and Integration (UDDI) is the service registry defined by the standard bodies for SOAP-based web services. However, it is rarely used in practice.

REST [2] uses a client-server architecture. REST does not restrict client-server communication to a particular protocol, but is most commonly used with HTTP because HTTP is the primary web transfer protocol. We therefore focus on using HTTP in this paper. RESTful web services can be described using the Web Application Description Language (WADL). A WADL file describes the requests that can legitimately be addressed to a service, including the service's Uniform Resource Identifier (URI) and the data the service expects and serves. REST relies on three main design principles: addressability, uniform interface, and statelessness. For addressability, REST models the data-sets to operate on as resources, and identifies each resource via a URI. A resource is any form of information that can be named and that is important enough to be referenced (e.g. a document, a row in a database, a search result). HTTP-based REST resources are accessible via the HTTP standard and uniform interface. Four main operations are supported: create, read, update and delete (CRUD). These operations can be implemented using the following four HTTP methods: POST, GET, PUT, and DELETE, although there is no one-to-one mapping between the methods and the operations. Regarding statelessness, each REST request is self-contained with all the information that the server needs to perform the requested action. The server never relies on information from previous requests to answer a new request.

This paper proposes a case study on a comparison of SOAP-based web services with their RESTful counterparts for the development of multimedia conferencing applications. Several aspects of multimedia conferencing have been standardized, including a SOAP-based web service programmatic interface (known as Parlay-X conferencing service [3]) that we re-use in this case study. The underlying model of Parlay-X conferencing service is based on three entities: conference, participant and media. The conference is the uniquely-identified context, to which participants can be added and removed. The participant is any party that participates in the conference. The media represents the media stream to support a participant's communication (e.g. audio, video, chat) and the stream direction (i.e. in, out, bidirectional).

For the case study, we define a RESTful conferencing service as a counterpart of the Parlay-X conferencing service. It offers the same functionality as the Parlay-X conferencing service, and is an extension of a simple RESTful conferencing service we described in a recent survey paper [4]. The resources are: a list of conferences, an individual conference, a list of participants, an

TABLE I. OUR CONFERENCING SERVICE RESOURCES

Resources	URL	HTTP action
List of conferences	http://conference.com/	POST: create a new conference
Individual conference	http://conference.com/{confId}	GET: return information about an individual conference DELETE: end an individual conference
List of participants	http://conference/{confId}/participants/	GET: return the list of an individual conference participants POST: invite a participant
Individual participant	http://conference.com/{confId}/participants/{participantURI}	GET: return information about a specific participant DELETE: disconnect an individual participant
Participant media	http://conference.com/{confId}/participants/{participantURI}/{media}	PUT: Add media for an individual participant DELETE: Delete media for an individual participant

individual participant, and participant media. Figure 1 summarizes their description. The next section is devoted to the system architecture used in the case study. It is followed by the implementation. We conclude with a brief summary and the lessons learned.

II. SYSTEM ARCHITECTURE

This section is organized as follows. We first present the overall system architecture, followed by the main interfaces. The last section is devoted to two illustrative scenarios.

A. OVERALL ARCHITECTURE

Figure 1 depicts the overall architecture we used in the case study. The conferencing gateway offers and implements SOAP-based web services' and RESTful web services' APIs for conferencing applications. These APIs can be used by different application server (AS) owners to develop new conferencing applications. In the case study, we used these APIs to develop a SOAP-based and a RESTful conferencing application. These applications can be used to create new multimedia conferences,

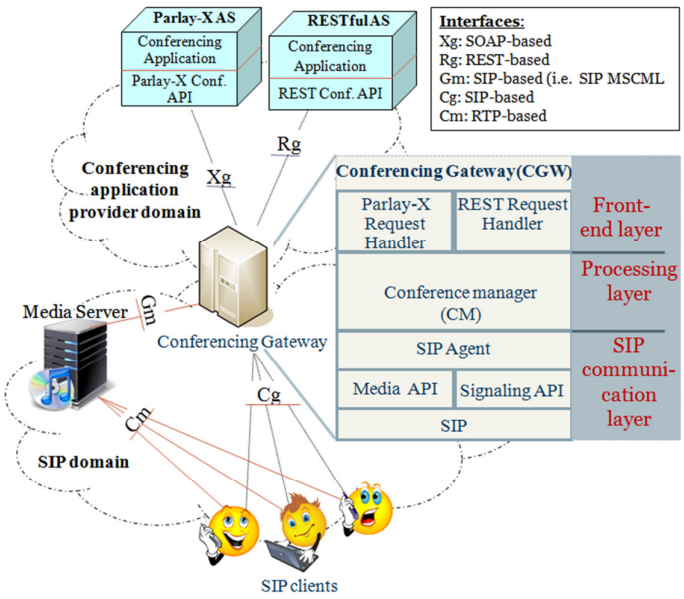

Figure 1: Overall system architecture

add/remove a participant to/from a conference, update a participant's media (e.g. change from audio to video), and delete a conference. The media communications between the conference participants are managed by a media server, which is controlled via the conferencing gateway. The case study is conducted in a SIP [5] environment. We chose SIP because it is the most widely deployed signalling protocol for multimedia conferencing.

The conferencing gateway architecture is composed of three layers: front-end, processing, and SIP communication layers. The front-end layer exposes the network conferencing capability towards the application servers. It includes two modules: Parlay-X request handler and REST request handler. The Parlay-X request handler manages SOAP communications with the Parlay-X applications, via the Rg interface. It receives SOAP conferencing requests from the applications, analyses them, and then passes their content (e.g. the method to be executed and its parameters) to the conference manager module in the processing layer. It also creates and sends SOAP responses to the Parlay-X application. The REST request handler module defines the REST interface of the conferencing gateway. It provides the same functionalities as the Parlay-X request handler module, but towards REST conferencing applications.

The processing layer contains the conferencing manager, which is responsible for the creation and management of the different multimedia conferences. As an illustration, let us examine how a SOAP /REST request for a conference creation is handled. The request is received by the Parlay-X/REST request handler; the handler then gets the method name (i.e. createConference) and parameters (e.g. the maximum number of participants) and passes them to the conference manager. With a SOAP request, the method name is included in the request body. In the case of a REST request, the method to be executed is identified via a correlation of the HTTP method in the request and the target resource (i.e. a POST request sent to the 'list of conferences' resource indicates a request to create a new conference). The conference manager communicates with the media server to create a new conference and to reserve the appropriate resources, and then replies back to the handler.

The SIP communication layer includes a SIP agent module that handles SIP exchanges between the conferencing gateway and the other entities in the network (i.e. the media server and the SIP end-users). The SIP agent is supported by two types of APIs: media and signalling APIs. The SIP agent creates and sends the appropriate SIP messages (e.g. SIP INVIT) to the appropriate SIP entities. It also receives and handles the SIP responses.

B. INTERFACES

The conferencing gateway provides two types of interfaces towards the application servers: Xg and Rg. Xg is SOAP-based and it offers the conferencing functionalities defined by the Parlay-X conferencing service. Rg is REST-based and a summary of its provided functionalities was presented in Table 1.

The conferencing gateway communicates with the media server via a SIP-based interface (i.e. Gm). The media server control protocol used at this interface is the SIP media server control markup language (SIP MSCML [6]), a language used in conjunction with SIP to provide advanced conferencing and interactive voice response functions.

The Cg interface between the conferencing gateway and the end-users is SIP based, and the Cm interface between the end-users and the media server is based on the Real-time Transport Protocol (RTP).

The rest of this section discusses the APIs supported by the processing and the SIP communication layers of the gateway architecture. The processing layer interfaces are presented first, followed by those of the SIP communication layer.

Processing layer interfaces: The processing layer defines three main APIs: *ConferenceManager*, *Conference*, and *Participant*. The *ConferenceManager* allows an application to create a new multimedia conference, or end an existing one. It also enables the application to check if a conference with a given identifier is ongoing (i.e. has been created and has not yet been terminated). The *Conference* API provides services for conference management, i.e. getInfo, getParticipants, addParticipant, and removeParticipant. The *Participant* API can be used to get participant information, or add/remove media to/from a given participant.

SIP communication layer interfaces: The main API at the SIP communication layer is *SIPAgent*. *SIPAgent* offers a set of services to send and receive SIP requests and responses. A separate method is defined to send each of the SIP requests (e.g. sendInvite), and a single method is used to send the different responses (i.e. sendResponse). The sendResponse method takes the code and the content of the response to send as parameters, e.g. sendResponse(200, null) is used to send a 200 OK response with no content. *SIPAgent* also provides services to process incoming requests (i.e. processRequest) and responses (i.e. processResponse).

C. ILLUSTRATIVE SCENARIOS

This section presents two illustrative scenarios: create conference and add participant.

Create conference: Figure 2 shows a sample sequence diagram to create an empty dial-out multimedia conference via both REST and SOAP interfaces. In a REST interface event, the RESTful conferencing application sends a POST request to the conferencing gateway, along with the conference information (e.g. maximum number of participants and conference duration) required to create a new conference (step 1). The request is first received and validated by the REST request handler, which

then sends a 202 Accepted response message to the application (step 2). We use a 202 status because even thought the request is accepted, the conference resource is not yet created (nor might it ever be, depending on the result of the communication with the media server). Next, the request handler passes the request content to the conference manager by calling the appropriate API, i.e. confManager.createConference(...) (step 3).

The conference manager creates a new conference object (with the conference information) which it stores locally, and then uses the SIP agent to send a SIP INVITE message to the media server to reserve resources for the new conference (steps 4 and 5). The media server response is then forwarded back to the conferencing application (steps 7, 9, 10 and 11).

A conference creation via the SOAP interface proceeds in a similar fashion, except that the communication between the SOAP-based conferencing application and the conferencing gateway passes through the Parlay-X request handler, and it is SOAP-based.

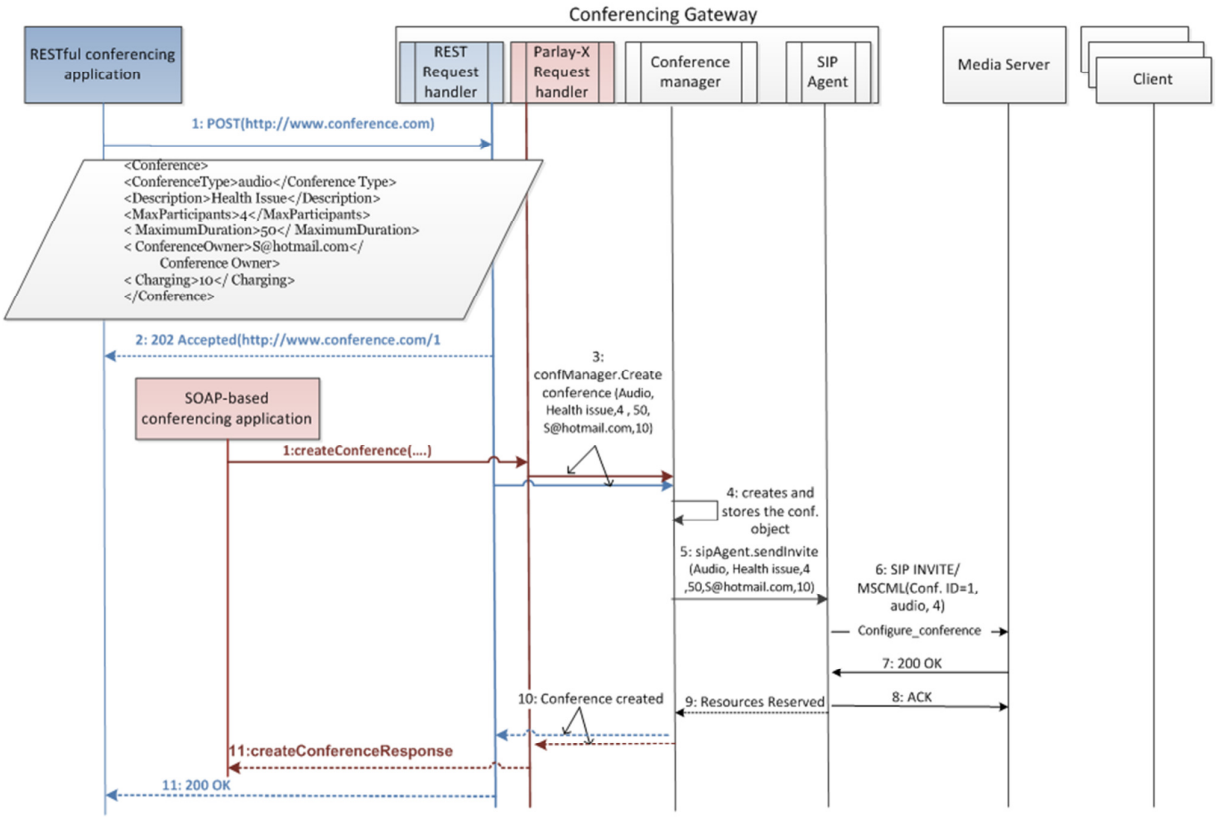

Figure 2: Create conference sequence diagram

Add participant: Figure 3 illustrates a sequence diagram to add a new participant (i.e. alice@sip.com) to the already created multimedia conference. Using the REST interface, the REST conferencing application sends a POST request to the conferencing gateway, with the URI of the participant and the identifier of the conference to which the participant should be added (i.e. 1). The request is received by the REST request handler, which verifies that the target conference exists, checks the request syntax, and replies with a 202 Accepted response (steps 2, 3, 4, and 5). We use 202 because at this stage, the participant is not yet added to the conference. The add participant request is rejected if the conference does not exist or if it has already reached the maximum number of participants (specified in the conference creation request).

In step 6, the REST request handler instructs the conference manager to add the participant to the identified conference. The conference manager then uses the SIP agent to invite the participant (step 7). The SIP agent moderates the negotiation of the session description information (e.g. IP address, media codec and port number) between the participant and the media server (steps 8-to-13). An RTP connexion is then created between the media server and the participant, and the conferencing application is notified that the participant has been added to the conference (steps 14, 15 and 16).

As in the create conference scenario, the SOAP-based conferencing application adds a participant following the same steps, excepts that the application will be communicating with the Parlay-X request handler and the message exchanges between the application and the handler will be SOAP-based.

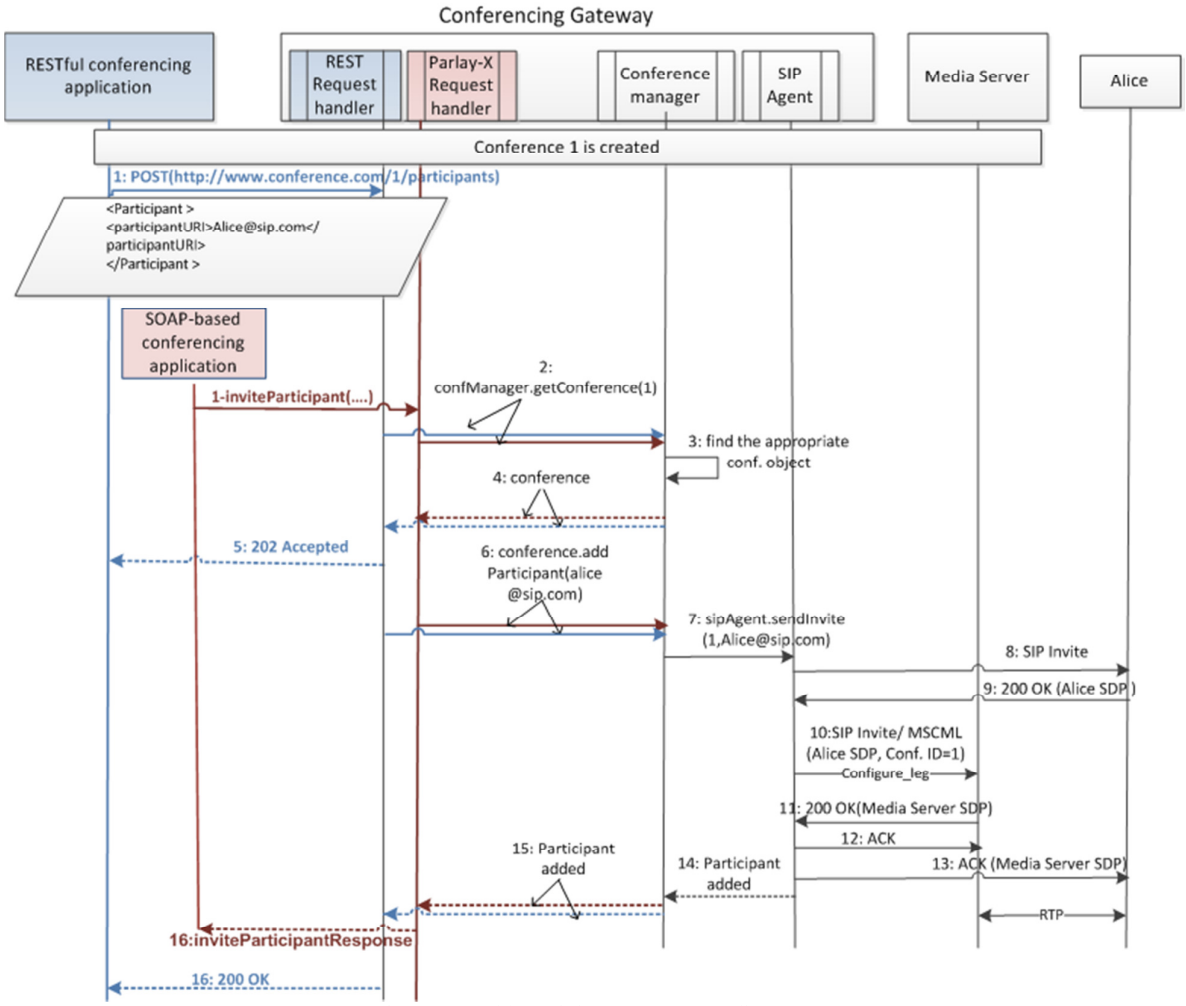

Figure 3: Add participant sequence diagram

III. IMPLEMENTATION

We present the implementation architecture first, then the description of the implemented prototype, and finally the performance measurements.

A. IMPLEMENTATION ARCHITECTURE

Figure 4 depicts the implementation architecture. The Parlay-X request handler is based on a SOAP API provided by the Oracle Enterprise Pack for Eclipse (OEPE), a set of Eclipse plug-ins designed to support application development for Oracle WebLogic application server. The REST request handler is based on Jersey API [7], an open source reference implementation of JSR 311 [8] for building RESTful web services.

The conference manager is composed of three modules: request dispatcher, conference management agent, and a database. The request dispatcher gets conference creation requests from the REST/Parlay-X request handler, creates a new conference

management agent, and gives it the responsibility to create a new conference. Each conference is managed by a separate conference management agent. The request dispatcher dispatches the subsequent requests to the appropriate agent. The requests related to a given conference are forwarded to the agent that created that conference. The relationships between conferences and their agents are preserved when the conferences are created. The management agents store the conference information in a local database. Such information includes, for instance, the unique conference identifier, the conference type (i.e., audio, video, chat etc.), the conference status (i.e., initiated, active and terminated), as well as the participants' information (e.g. number of participants, participants' URIs, and type of media for each participant).

The SIP agent module is implemented as a SIP servlet [9], which is deployed on a SIP servlet engine. The SIP servlet creates and sends the actual SIP messages (e.g. SIP INVITE, SIP ACK) and handles the received SIP responses. JSR 289 [9] is used to provide the required SIP APIs.

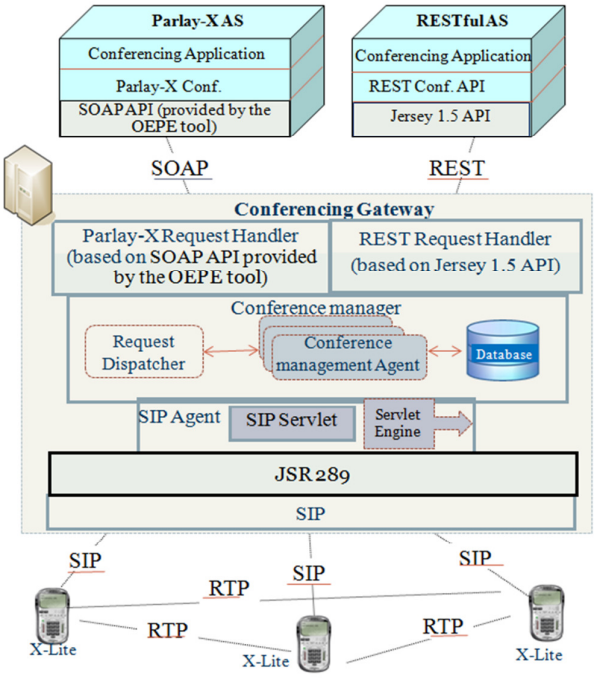

Figure 4: Implementation architecture

B. PROTOTYPE

A proof-of-concept prototype is implemented and tested using various scenarios. The prototype includes a Parlay-X-based conferencing application, a REST-based conferencing application, a conferencing gateway, and a set of SIP clients. No media server is used for the prototype. The implemented conferencing dial-out applications are distributed. The clients are connected in a full-mesh topology from the media handling perspective and each client does its own mixing.

Both applications are developed as web applications, using an Eclipse environment and OEPE. They are deployed on Oracle Weblogic Server 11gR1, an application server that supports natively both SOAP and REST applications. The conferencing gateway is fully implemented, and is also deployed on Oracle Weblogic Server 11gR1. X-lite soft phones are used as SIP clients.

Our prototype is used to build empty dial-out multimedia conferences and then to add participants one by one (following Parlay-X conferencing service specifications). Different conferencing operations were tested, including add participant, remove participant, end conference, get conference participants, get a participant's information, and get conference information.

C. PERFORMANCE MEASUREMENTS

We first describe the experiment setup, and then we present the performance metrics used. The performance results are presented and discussed at the end.

Experiment setup: The experiment is set up with one Parlay-X application, one REST application, one conferencing gateway, and some clients. To get a fair comparison between the two applications, both are run on the same laptop (with an i3 processor, 3 GB RAM, and Windows 7). However, only one application is executed at a time for measurement purposes. The same laptop also runs a SIP client.

The conferencing gateway is deployed on a second laptop, equipped with a dual core processor, 4 GB RAM, running Windows Vista. Two additional laptops are used for two other SIP clients. These laptops are configured with dual core processors, 1 GB RAM, and Windows 7.

Metrics: The performance of the prototype is evaluated in terms of the end-to-end time delay and network load when executing different conferencing application operations. The delays are measured as the difference between the time when the conferencing application (i.e. either SOAP-based or RESTful) sends a request and the time it receives a response from the conferencing gateway. The time for inviting a participant, for example, includes: the time to send a request to the gateway, the time to send an INVITE message to the participant and to get its acceptance, and the time to send the corresponding response to the application. Network load indicates the total number of bytes sent and received by conferencing applications (when communicating with the conferencing gateway) for a given request. The same data model and format were used for both applications; whenever possible (e.g. the RESTful request to get a conference information has an empty body, which is not the case of the counterpart SOAP request).

We also measure the same delays when all of the components are running on the same machine. This eliminates the delays induced by messages' delivery over the network and gives a good idea about the processing delays for both SOAP-based and RESTful requests.

Performance results: Table 2 shows the evaluation results. The delays are measured in milliseconds, and each result is calculated as the average of 10 experiments. Table 2 compares the delays via SOAP and REST interfaces. The delays incurred

TABLE II. PERFORMANCE RESULTS

	SOAP-based			REST-based		
Multimedia Conferencing API	**Delay in a distributed environment**	**Delay on the same machine**	**Network load**	**Delay in a distributed environment**	**Delay on the same machine**	**Network load**
Create conference	848.4	381.7	767	171.4	102.7	273
Get conference information	818.6	335.3	546	172.3	98.6	177
Add participant	1325.3	334.2	578	368.8	103.3	200
Remove participant	1322.3	357	588	382.9	107.2	195
Get participants	787.1	342.7	615	167.8	104.8	195
Get participant information	766.2	346.7	619	169.8	105	204
End conference	1508.4	341.4	500	556.6	105.3	204

with a REST-based interface in a distributed environment are three to five times less than when using the SOAP-based interface. Both applications are run on the same environment. The only difference between the development and deployment of the two applications lies in the use of SOAP. Indeed, both applications have the same web interface, and the differences at the conferencing gateway side are limited to the Parlay-X and REST request handler modules. We can therefore safely conclude

that the difference in delays is due to the use of SOAP. In SOAP-based web services, the requests are written in a SOAP format, than enveloped in an HTTP message. RESTful web services do not use SOAP or any other application layer protocol other than HTTP.

The delay differences when a single machine is used follow a similar pattern as in a distributed environment (i.e. REST delays are three to four times less than their SOAP counterparts). This is basically due to the SOAP messages processing (e.g. open the envelope and extract the name of the target service as well as the name and the parameters of the method to be executed). In REST requests, the target service and method are given in the request URI. The method parameters (if present) still have to be extracted from the request body.

Our findings support the quantitative measurement results published in [10], which show that the processing time of SOAP-based web service requests in a mobile environment may be ten times longer than the processing time of an equivalent RESTful web service request, and may consume eight times more memory. The difference between these findings and ours may be explained by the fact that mobile devices are resource-constrained (and therefore more processing time is required), in contrast with the laptops used in our experiments.

The differences between the execution delays in a distributed environment and in a single machine approximate the network delays for transferring messages through the network. These differences are two to four times higher for SOAP requests. This can be explained by the overhead induced by SOAP, in termes of network load. This overhead is due to the mandatory SOAP body (e.g. a SOAP body needs to be sent to get a conference information whereas the body of the counterpart RESTful request is empty) and the extra information added in SOAP messages (e.g. SOAP envelope). The network load overhead is two to three times less when using a REST interface.

IV. RELATED WORK

Our work contrasts SOAP-based web service programmatic interfaces with their RESTFul web service counterparts in a concrete case study using a multimedia conferencing application. The case study includes the prototyping of identical multimedia conferencing applications using the two programmatic interfaces. There are few papers that contrast SOAP-based web services programmatic interfaces with their RESTFul counterparts. Reference [11] is one example. It provides a technical comparison of the two programmatic interfaces. It considers architectural principles, conceptual decisions, and technology decisions. From an architectural principles standpoint, it compares how the protocols are layered, how the two interfaces deal with heterogeneity, and how they define loose coupling. The comparison of conceptual decisions focuses on the differences between the integration styles subjacent to the two interfaces. Transport protocols, payload format, service discovery and service composition are examples of the criteria used for the comparison from the technology viewpoint. The paper does provide a number of insights as far as the theoretical comparison between the two interfaces. Unfortunately, it provides little practical insight because it does not focus on concrete case studies that include prototypes and measurements. Reference [12] is another example. The scope of the comparison is restricted to the choreography standards that come with the two interfaces. The standardization processes, the standard specifications and also the public comments on the specifications are considered. The study provides a number of insights that are of obvious interest to standardization stakeholders. Unfortunately, it provides no practical information on the differences between the two interfaces in terms of concrete application development.

There are several papers that deal with multimedia conferencing application development with SOAP-based web services programmatic interfaces. However, they usually do not attempt to contrast this with the development of the same applications using RESTFul web services counterparts. Reference [13] addresses the general issue of SOAP-based web service multimedia conferencing in the IP Multimedia Subsystem (IMS). IMS is the service delivery platform of next generation telecommunications networks. The two programmatic interfaces supported by IMS are discussed, namely the OSA conference management API, and the SOAP-based Parlay-X conferencing interface used in this paper. The shortcomings are discussed and the paper argues that the most promising solution is the extension of the SOAP-based Parlay-X conferencing interface. Reference [14] proposes a SOAP-based web service interface for multimedia conferencing. That interface predates the standardization of the SOAP-based web services used in this paper. The authors provide a mapping of the interface onto the OSA conference management API and provide their preliminary measurements. They also discuss the prototype they built in a SIP environment. Reference [15] deals with sessions-based applications at large, in converged communication services settings. It does not focus on conferencing applications. It proposes a SOAP-based web service interface that enables a client in a session to act as a SOAP endpoint. The client can receive and parse SOAP messages that are sent from a centralized application. The same client can also act as a SIP end-point. Reference [16] also deals with sessions-based applications at large, with no focus

on conferencing applications. It proposes a general SOAP-based web services framework for real time communications and converged services over IP. The framework includes SOAP-based web services for session management applications.

Some papers have investigated the use of RESTful web services for multimedia conferencing applications, but with no attempt at a comparison with the SOAP-based web services counterparts. Reference [17] is one example. It focuses on IMS and aims at the convergence of telecommunications networks and Web 2.0. A reference model is proposed and an embryonic RESTful conferencing model is sketched. The model includes the state, list of participants, media description and a link for media components as resources. However, the model has not been implemented and no concrete prototype is described. Several other papers have researched the use of RESTful web services as programmatic interfaces in IMS, but for applications other than multimedia conferencing. One example is reference [18]. It focuses on instant messaging and presence service. The interfaces are specified and the prototype described. Another example is provided by reference [19], which also focuses on presence. An IMS application with avatars as mash up parts for blogs, SMSs and schedule systems is prototyped as proof of concept. Yet another example is reference [20], which deals with streaming services. The focus is on user-generated video content, and a framework for video streaming over IMS is proposed. A prototype, developed with a RESTful web service interface, makes the framework available to developers.

V. CONCLUSIONS

In this paper, we present a case study on a comparison of SOAP-based web services with their RESTful counterparts for the development of multimedia conferencing applications. Two reference applications are used for the case study: a SOAP-based and a RESTful multimedia conferencing application. The SOAP-based application is an implementation of the Parlay-X conferencing service. The RESTful application implements a RESTful conferencing service we defined as a counterpart of the Parlay-X conferencing service and which offers the same functionality.

The applications can be used to create new multimedia conferences, to add a participant to or remove a participant from an ongoing conference, to update a participant's media (e.g. change from audio to video), and to delete a conference. Both applications are developed in the same environment and deployed on the same application server. Various scenarios are run and performance measurements are taken.

The main lesson we learned from this case study is related to the end-to-end delays for executing the different conferencing operations (e.g. create conference, add participant). The delays via the REST interface are three to five times less than when using SOAP; both applications having been run on the same environment. The difference is due to the use of SOAP. This assessment is in line with previous published results which showed that the processing time of SOAP-based web service requests in a mobile environment may be ten times higher than the processing of an equivalent RESTful web service request.

A second lesson concerns the selection of the application server. For the implementation and deployment of the conferencing gateway, we needed an application server that supports both RESTful and SOAP-based web services, as well as SIP. We had to review various existing development tools and application servers, and discovered that Oracle Weblogic Server 11gR1 meets all three requirements.

ACKNOWLEDGMENTS

This work was supported in part by funding from the Canada Research Chair in End-User Service Engineering for Communications Networks.

REFERENCES

[1] E. Newcomer, Understanding Web Services: XML, WSDL, SOAP, and UDDI, Addison-Wesley, May 2002
[2] Fielding, Roy Thomas. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000
[3] ETSI TS 129 199-12, "Parlay X Web Services; Part 12: Multimedia Conference," 3GPP TS 29.199-12 v. 9.0.0, Rel. 9, Jan. 2010
[4] F. Belqasmi et al, RESTful Web Services for Service Provisioning in Next Generation Networks: A Survey, IEEE Communications Magazine, vol.49, no.12, pp.66-73, December 2011
[5] Rosenberg et al. 'SIP: Session Initiation Protocol', RFC3261, June 2002.
[6] J. Van Dyke et al., "Media Server Control Markup Language (MSCML) and Protocol," RFC 5022, Nov. 2007

[7] Jersey API, version 1.11; available on web at: http://jersey.java.net/; February 2012
[8] M. Hadley and P. Sandoz; JAX-RS: Java™ API for RESTful Web Services, Version 1.1, September 2009
[9] M. Kulkarni et al; SIP Servlet Specification, version 1.1; JSR 289 Expert Group; August 2008
[10] F. AlShahwan, K. Moessner, "Providing SOAP Web Services and RESTful Web Services from Mobile Hosts" 2010 Fifth International Conference on Internet and Web Applications and Services (ICIW), pp.174-179, 2010
[11] C. Pautasso et al., RESTful Web Services vs. Big Web Services : Making the Right Architectural Decision, WWW 2008, Beijing China
[12] M. Muehlen et al., Developing Web Services Choreography Standards – The Case REST vs. SOAP, Elsevier Decision Support Systems 40 (2005) 9-29
[13] M. Gomez and T. de Miguel, Advanced IMS Multipoint Conference Management Using Web Services, IEEE Communications Magazine, July 2007
[14] J. Torreira et al., Web Services in 3G Networks: A Parlay Based – Implementation, ICN 2004, Bordeaux, France
[15] F. Liu et al., WSIP – Web Service SIP endpoint for Converged Multimedia / Multimodal Communication Over IP, IEEE International Conference on Web Services, 2004
[16] W. Chou et al, Web Services for Communications over IP, IEEE Communications Magazine, March 2008
[17] D Lozano et al, WIMS 2.0: Converging IMS and Web 2.0 – Designing REST APIs for the exposure of session based – IMS capabilities, Second International Conference on Next Generation Mobile Applications, Services and Technologies, 2008
[18] H.M Rissanen et al., Design and Implementation of a RESTful IMS API, Sixth International Conference on Wireless and Mobile Communications, 2010
[19] N Takaya et al, Presence with avatar for Web 2.0 – IMS Services using REST Interfaces, 12th International Conference on Intelligence in Next Generation Networks, 2008 (ICIN 2008),
[20] Devdutt et al, A Framework for Converged Video Services in the IP Multimedia Subsystem, IMSAA 2009, December 9 – 11, Bangalore, India

Biographies

Fatna Belqasmi holds a Ph.D. and an M.Sc. degree in electrical and computer engineering from Concordia University, Canada. She is a research associate at Concordia University, Canada. In the past, she worked as a researcher at Ericsson Canada. She was part of the IST Ambient Network project (a research project sponsored by the European Commission within the Sixth Framework Programme -FP6-). She worked as an R&D engineer for Maroc Telecom in Morocco. Her research interests include next generation networks, service engineering, distributed systems, and networking technologies for emerging economies.

Jagdeep Singh (jagd_si@encs.concordia.ca) is currently working on an M.Sc. in electrical and computer engineering at Concordia University. He holds a B.Eng degree in electronics and communication engineering from Thapar University, India. From 2008 to 2010, he worked as an Assistant System Engineer in Telecom domain at Tata Consultancy Services (TCS), India. His industrial experience includes software development and network testing. His research interests include web services, services for next generation networks, and IMS.

Suhib Younis Bani Melhem (s_banim@encs.concordia.ca) holds an M.Eng. degree in electrical and computer engineering from Concordia University, Canada, and a Bachelor and an M.Eng. degree in computer engineering from Jordan University of Science and Technology, Jordan. He is currently a Ph.D. student at Concordia University and he is working on his thesis under the supervision of Dr. Roch Glitho. His research interests include distributed systems (RESTful Web Services), service engineering (Multimedia conferencing), and 4G networks.

Roch Glitho [SM] (Roch H. Glitho [SM] (http://www.ece.concordia.ca/~glitho/) holds a Ph.D. (Tekn. Dr.) in tele-informatics (Royal Institute of Technology, Stockholm, Sweden) and three M.Sc. degrees: business economics (University of Grenoble, France), pure mathematics (University Geneva, Switzerland), and computer science (University of Geneva). He is an Associate Professor of Networking and Telecommunications at Concordia University where he holds the Canada Research Chair in End-User Service Engineering for Communications Networks, and leads the Telecommunications Service Engineering Laboratory (TSE Lab). In the past he has worked in industry for almost a quarter of a century and has held several senior technical positions at LM Ericsson in Sweden and Canada (e.g. expert, principal engineer, senior specialist). In the past he has served as IEEE Communications Society distinguished lecturer, Editor-In-Chief of IEEE Communications Magazine and Editor-In-Chief of IEEE Communications Surveys & Tutorials. His research areas include architectures for end-users services, distributed systems, non conventional networking, and networking technologies for emerging economies.