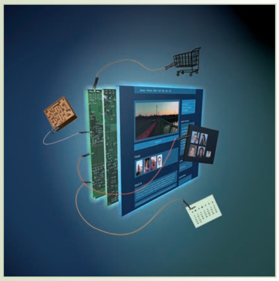


SOAP-Based vs. RESTful Web Services

A Case Study for Multimedia Conferencing

RESTful Web services are now emerging as an alternative to SOAP-based Web services and might be a more suitable choice in some cases. A comparison of two Web programming interfaces — the standard Parlay-X multimedia SOAP-based Web service and a RESTful Web service that offers the same functionalities — for developing multimedia conferencing applications shows that the RESTful Web interface offers better performance.

Fatna Belqasmi, Jagdeep Singh, Suhib Younis Bani Melhem, and Roch H. Glitho
Concordia University

Multimedia conferencing applications – such as audio/video conferencing, multiparty online games, and distance learning – are an important category of Web applications. Such applications are rapidly increasing in popularity, and developers are using them in different application areas. Like all Web applications, conferencing applications can be developed using Web services based either on SOAP[1] or on REST principles.[2] Researchers and developers are now considering RESTful Web services as an alternative that might be more adequate than SOAP-based Web services in some cases. RESTful Web services are deemed more lightweight because they don't use SOAP or any other application layer protocol except for HTTP.

Here, we offer a case study comparing SOAP-based Web services with their RESTful counterparts for developing multimedia conferencing applications. We reuse several standardized aspects of multimedia conferencing, including the Parlay-X conferencing service,[3] a SOAP-based Web service programmatic interface. Parlay-X's underlying model 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 (such as audio, video, or chat) and the stream direction (that is, in, out, or 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 service, and extends a simple RESTful conferencing service we recently described elsewhere.[4]

Overview of the Service Architectures

The SOAP-based Web service architecture comprises three entities:

- a service provider, which creates a SOAP-based Web service and publishes the service description in the service registry;
- a service registry, which enables online service discovery; and
- a service requestor, which finds the service by querying the service registry. The requestor then retrieves the service description, uses it to bind to the service implementation, and begins interacting with it.

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), which provides information on how to use a Web service, including a description of the service methods and binding information. UDDI is the service registry defined by the standard bodies for SOAP-based Web services. However, it's rarely used in practice.

REST uses a client-server architecture. REST doesn't restrict client-server communication to a particular protocol, but is most commonly used with HTTP. We therefore focus on HTTP use here. RESTful Web services can be described using the Web Application Description Language. A WADL file describes the requests that can legitimately be addressed to a service, including the service's 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 datasets 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's important enough to be referenced (such as a document, a row in a database, or 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), which are implemented using `POST`, `GET`, `PUT`, and `DELETE`, respectively. There is no strict mapping in the opposite direction — for example, `PUT` can also be used to create new resources, not just to update. Regarding statelessness, each REST request contains all the information the server needs to perform the requested action. The server never relies on information from previous requests to answer a new one.

Case Study Architecture

Figure 1 shows our case study architecture. The conferencing gateway offers and implements SOAP-based and RESTful Web services' APIs for conferencing applications. Different application server (AS) owners can use these APIs to develop new conferencing applications. In the case study, we used the APIs to develop a SOAP-based and a RESTful conferencing application. Users can employ these applications to create new multimedia conferences, add or remove a conference participant, update a participant's media (such as change from audio to video), and delete a conference. The media communications between conference participants are managed by a media server, which is controlled via the conferencing gateway. The case study is conducted in a Session Initiation Protocol (SIP)[5] environment. We chose SIP because it's the most widely deployed signaling 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 to application servers. It includes two modules: the Parlay-X request handler and the 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 (such as 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 conferencing gateway's REST interface. It provides the same functionalities as the Parlay-X request handler module, but offers them to REST conferencing applications.

The *processing layer* contains the conferencing manager, which creates and manages

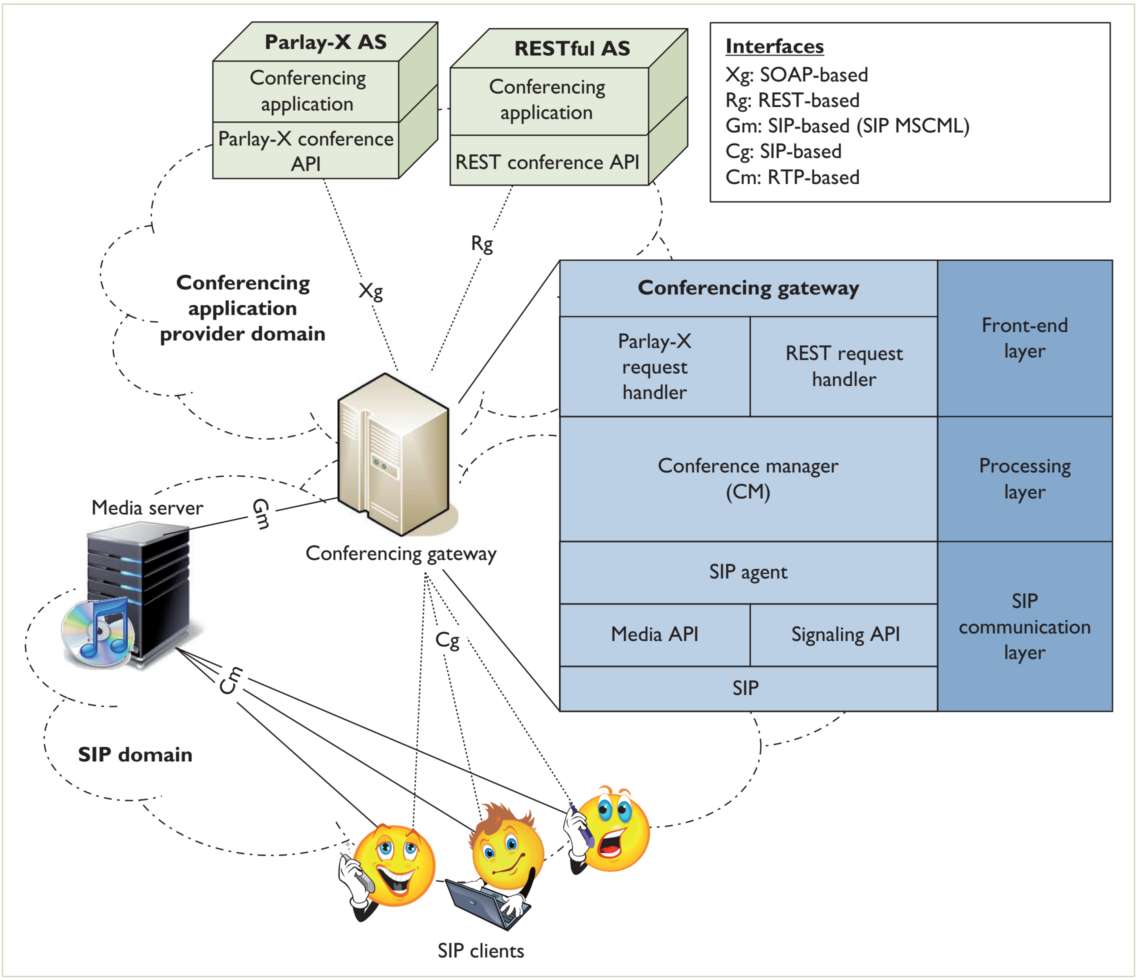


Figure 1. Overall system architecture. The conferencing gateway exposes the network conferencing capability to application developers via two interfaces: a SOAP-based interface and a RESTful interface.

the different multimedia conferences. A SOAP/REST request for a conference creation, for example, entails the following steps:

- The Parlay-X/REST request handler receives the request.
- The handler then gets the method name (that is, `createConference`) and parameters (such as the maximum number of participants). With a SOAP request, the method name is included in the request body. With a REST request, the handler identifies the method to be executed by correlating the request's HTTP method and the target resource (that is, a `POST` request sent to the "list of conferences" resource indicates a request to create a new conference).
- The handler passes the method name and parameters to the conference manager. The conference manager communicates with the media server to create a new conference and reserve the appropriate resources, and then replies to the handler.

The SIP communication layer includes a SIP agent module that handles SIP exchanges between the conferencing gateway and other network entities (the media server and SIP end users). Two types of APIs support the SIP agent: media and signaling APIs. The SIP agent creates and sends the appropriate SIP messages (such as SIP `INVITE` as is used later) to the appropriate SIP entities. It also receives and handles the SIP responses.

Interfaces

The conferencing gateway provides two types of interfaces toward the application servers: Xg

Table 1. 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.com/{confId}/participants/	`GET`: return the list of 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

and Rg. Xg is SOAP-based and offers the conferencing functionalities defined by the Parlay-X conferencing service. Rg is REST-based; Table 1 summarizes its provided functionalities.

The conferencing gateway communicates with the media server via a SIP-based interface (that is, Gm). The media server control protocol used at this interface is the SIP Media Server Control Markup Language (SIP MSCML),[6] which is used in conjunction with SIP to provide advanced conferencing and interactive voice response functions.

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

Processing layer interfaces. The processing layer defines three main APIs: `ConferenceManager`, `Conference`, and `Participant`. The `ConferenceManager` lets an application create a new multimedia conference or end an existing one. It also lets the application check whether a conference with a given identifier is ongoing (that is, it's been created and isn't yet terminated). The `Conference` API provides conference management services — that is, `getInfo`, `getParticipants`, `addParticipant`, and `removeParticipant`. The `Participant` API can be used to get participant information or add or remove media for a given participant.

SIP communication layer interfaces. The main API at the SIP communication layer is `SIPAgent`, which 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 (such as `sendInvite`), and a single method — `sendReponse` — sends the different responses. The `sendResponse` method takes the response's code and content as parameters; for example, `sendResponse(200, null)` is used to send a `200 OK` response with no content. `SIPAgent` also provides services to process incoming requests (`processRequest`) and responses (`processResponse`).

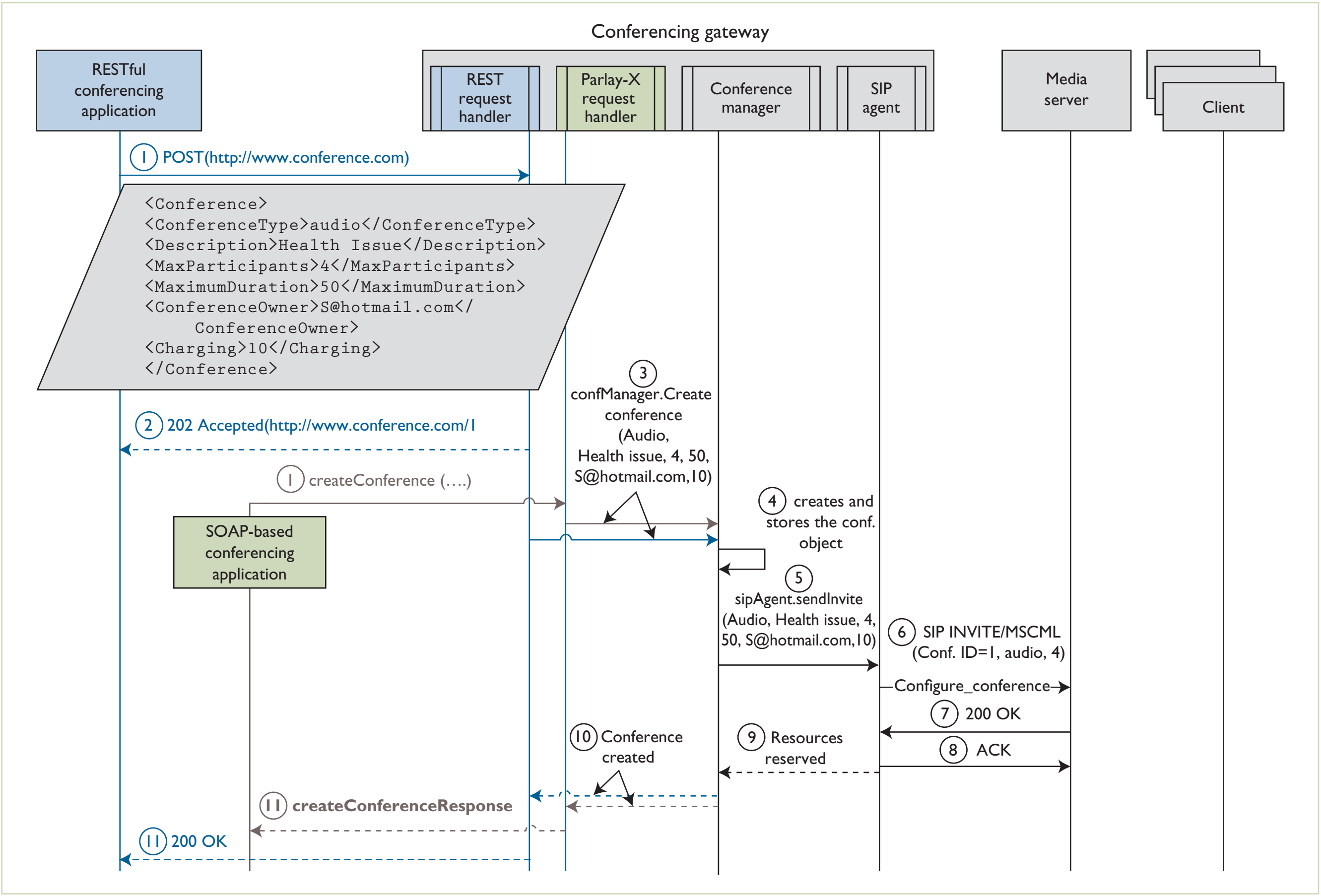
Example Scenarios

To illustrate how the architecture works, we describe two scenarios: create a conference and add a participant.

Create a 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 information — such as maximum 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, although the request is accepted, the conference resource isn't yet created (and might never be, depending on the result of communications with the media server). Next, the request handler passes the request content to the conference manager by calling the appropriate API — that is, `confManager.createConference(…)` (step 3).

The conference manager uses the received conference information (such as maximum participants) to create a new conference object and then stores the object in a local database. It then uses the SIP agent to send a SIP `INVITE` message


Conferencing gateway
RESTful conferencing application
REST request handler
Parlay-X request handler
Conference manager
SIP agent
Media server
Client
1 POST(http://www.conference.com)
2 202 Accepted(http://www.conference.com/1
1 createConference (….)
SOAP-based conferencing application
3 confManager.Create conference (Audio, Health issue, 4, 50, S@hotmail.com,10)
4 creates and stores the conf. object
5 sipAgent.sendInvite (Audio, Health issue, 4, 50, S@hotmail.com,10)
6 SIP INVITE/MSCML (Conf. ID=1, audio, 4)
Configure_conference
7 200 OK
8 ACK
9 Resources reserved
10 Conference created
11 createConferenceResponse
11 200 OK

```
<Conference>
<ConferenceType>audio</ConferenceType>
<Description>Health Issue</Description>
<MaxParticipants>4</MaxParticipants>
<MaximumDuration>50</MaximumDuration>
<ConferenceOwner>S@hotmail.com</
     ConferenceOwner>
<Charging>10</Charging>
</Conference>
```

Figure 2. Create conference sequence diagram. The SOAP-based and RESTful conferencing applications send a request to the gateway with a description of the conference to be created, and the gateway takes care of the actual creation.

to the media server to reserve resources for the new conference (steps 4–6). The media server response is forwarded back to the conferencing application (steps 7 and 9–11).

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 SOAP-based Parlay-X request handler.

Add a participant. Figure 3 shows a sequence diagram for our second scenario, in which we'll add a new participant (alice@sip.com) to the existing multimedia conference. Using the REST interface, the REST conferencing application sends a `POST` request to the conferencing gateway, with the participant's URI and the identifier of the conference to which the participant should be added (that is, step 1). The REST request handler receives the request, verifies that the target conference exists, checks the request syntax, and replies with a `202 Accepted` response (steps 2–5). 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 doesn't exist or has already reached its maximum number of participants (as 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 (such as IP address, media codec, and port number) between the participant and the media server (steps 8–13). An RTP connection 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 14–16).

The SOAP-based conferencing application adds a participant following the same steps as the RESTful application, except that the SOAP-based application communicates with the Parlay-X

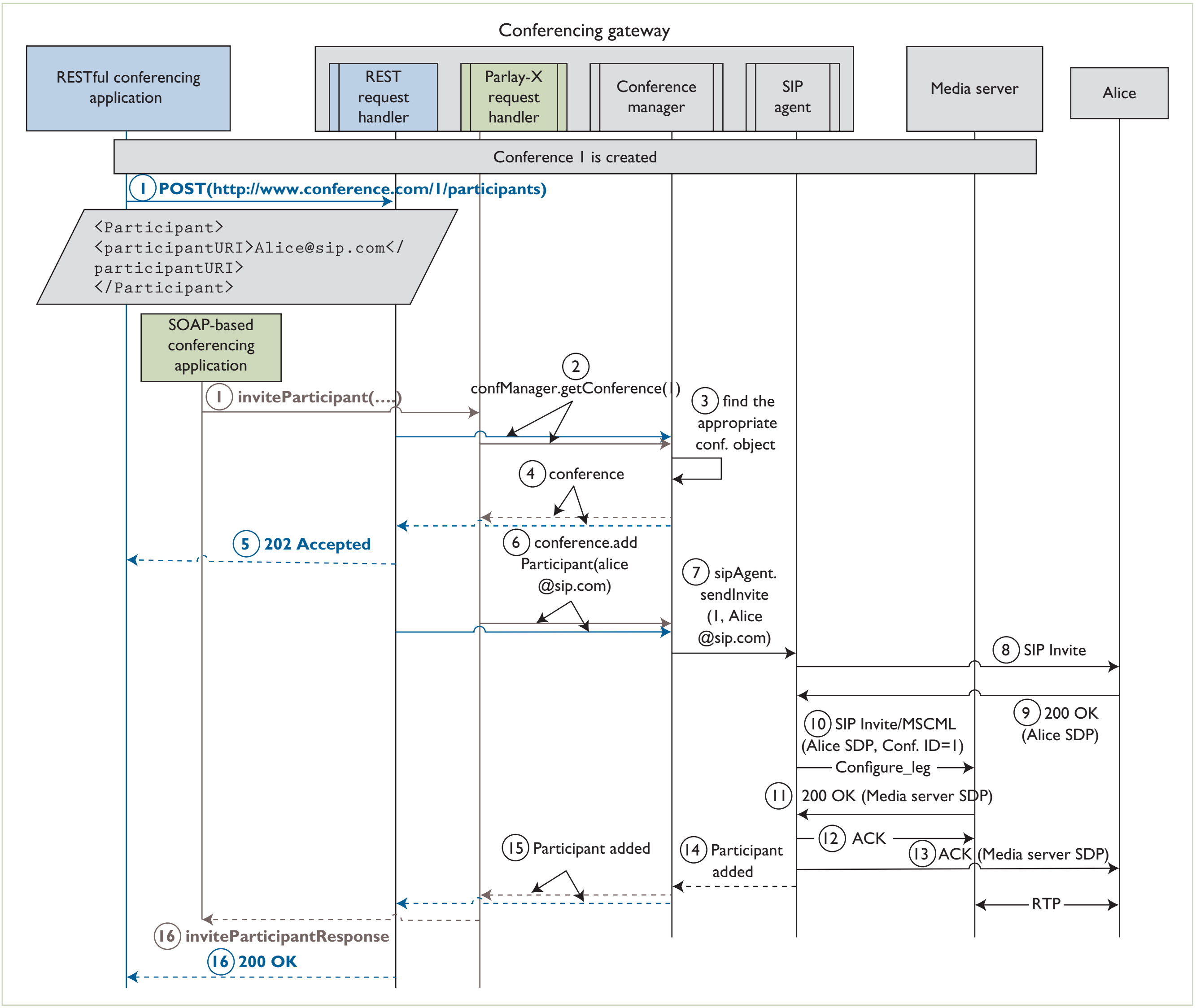

Conferencing gateway
RESTful conferencing application
REST request handler
Parlay-X request handler
Conference manager
SIP agent
Media server
Alice
Conference 1 is created
1 POST(http://www.conference.com/1/participants)
<Participant>
<participantURI>Alice@sip.com</participantURI>
</Participant>
SOAP-based conferencing application
1 inviteParticipant(….)
2 confManager.getConference(1)
3 find the appropriate conf. object
4 conference
5 202 Accepted
6 conference.addParticipant(alice@sip.com)
7 sipAgent.sendInvite(1, Alice@sip.com)
8 SIP Invite
9 200 OK (Alice SDP)
10 SIP Invite/MSCML (Alice SDP, Conf. ID=1) Configure_leg
11 200 OK (Media server SDP)
12 ACK
13 ACK (Media server SDP)
14 Participant added
15 Participant added
RTP
16 inviteParticipantResponse
16 200 OK

Figure 3. Add participant sequence diagram. The conferencing applications request adding Alice to an existing conference identified by its unique ID, and the gateway invites Alice and moderates the communication between Alice and the media server.

request handler, and the messages between the application and handler are SOAP-based.

Implementation

Figure 4 shows 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 the Oracle WebLogic application server. The REST request handler is based on the Jersey API (http://jersey.java.net), an open source reference implementation of Java Specification Request (JSR) 311[7] for building RESTful Web services.

The conference manager has three modules: request dispatcher, conference management agent, and a database. The *request dispatcher* receives 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. A separate *conference management agent* manages each conference. The request dispatcher dispatches the subsequent requests to the appropriate agent. It forwards the requests related to a given conference to the agent that created that conference. The relationships between conferences and their agents are preserved when the conferences are created. Finally, the management

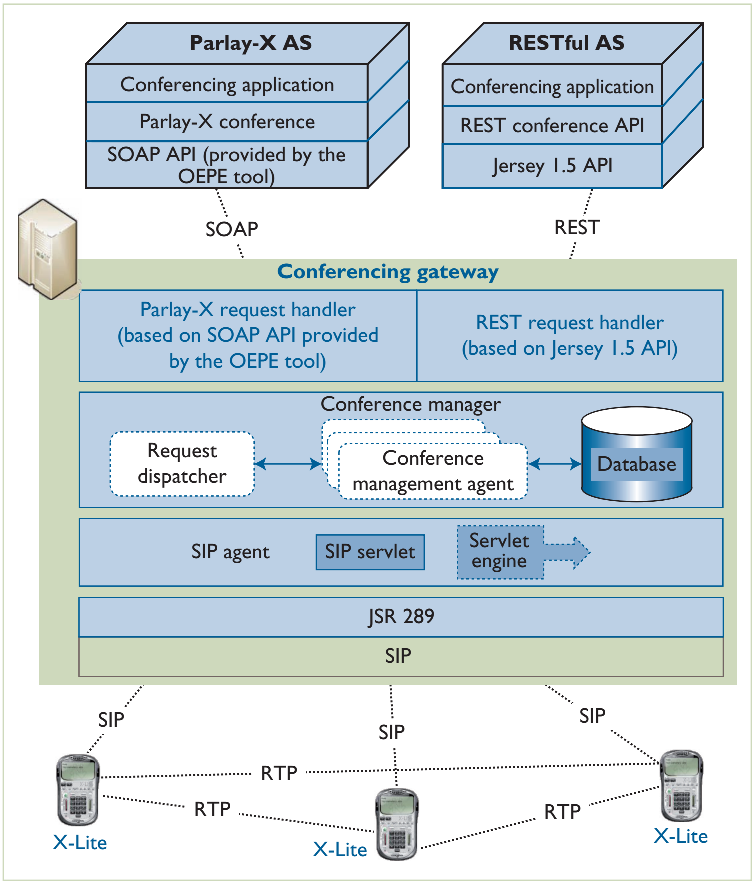


Figure 4. Implementation architecture. The SOAP-based and the RESTful conferencing applications use similar implementation architectures, except for the use of SOAP for communicating with the conferencing gateway.

agents store the conference information in a local database. This information includes the unique conference identifier, the conference type (such as audio, video, or chat), the conference status (initiated, active, or terminated), and participant information (including the number of participants, their URIs, and each participant's media type).

The SIP agent module is implemented as a SIP servlet,[8] which is deployed on a SIP servlet engine. The SIP servlet creates and sends the actual SIP messages (such as `SIP INVITE` and `SIP ACK`) and handles the received SIP responses. JSR 289[8] provides the required SIP APIs.

Prototype

We implemented a proof-of-concept prototype and tested it 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. The prototype doesn't use a media server. The implemented conferencing dial-out applications are distributed. Clients are connected in a full-mesh topology from the media-handling perspective, and each client does its own mixing.

We developed the two applications as Web applications using the Eclipse and OEPE environments. We deployed the applications on Oracle Weblogic Server 11gR1, which offers native support for both SOAP and REST applications. To implement and deploy the conferencing gateway, we needed an application server that supports both RESTful and SOAP-based Web services, as well as SIP. After reviewing various existing development tools and application servers, we discovered that Oracle Weblogic Server 11gR1 meets all three requirements; we can therefore fully implement and deploy the conferencing gateway on it. We used X-lite soft phones as SIP clients.

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

Performance Measurements

Our experiment used one Parlay-X application, one REST application, one conferencing gateway, and three clients. To achieve a fair comparison between the two applications, we ran both on the same laptop (with an i3 processor, 3 Gbytes of RAM, and Windows 7). However, we executed only one application at a time for measurement purposes. The same laptop also runs a SIP client.

We deployed the conferencing gateway on a second laptop running Windows Vista and equipped with a dual-core processor and 4 Gbytes of RAM. We used two additional laptops for two other SIP clients; these laptops are configured with dual-core processors, 1 Gbyte of RAM, and Windows 7.

Metrics. We evaluated prototype performance in terms of end-to-end time delay and network load when executing different conferencing application operations. We measured delays as the difference between the time when the

Table 2. Performance results.

	SOAP-based			REST-based		
Multimedia conferencing API	**Delay in a distributed environment (ms)**	**Delay on the same machine (ms)**	**Network load (bytes)**	**Delay in a distributed environment (ms)**	**Delay on the same machine (ms)**	**Network load (bytes)**
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

conferencing application 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 receive 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. We used the same data model and format for both applications whenever possible (for example, the RESTful conference information request has an empty body, which isn't the case in the SOAP-based counterpart).

We also measured the same delays when all components were running on the same machine. This eliminates the delays induced by message 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. We measured the delays in milliseconds and calculated each result as the average of 10 experiments. Table 2 compares the delays via SOAP and REST interfaces. The delays incurred with a REST-based interface in a distributed environment are three to five times less than with the SOAP-based interface. 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 SOAP. In SOAP-based Web services, the requests are written in a SOAP format, then enveloped in an HTTP message. RESTful Web services don't use SOAP or any other application-layer protocol other than HTTP.

The delay differences on a single machine are similar to those in a distributed environment (that is, REST delays are three to four times less than those with SOAP). This is basically due to the SOAP messages' processing, which, for example, opens the envelope and extracts the name of the target service as well as the name and parameters of the method to be executed. In contrast, in REST requests, the target service and method are given in the request URI and only the method parameters (if present) are extracted from the request body.

Our findings support quantitative measurement results published elsewhere,[9] which show that processing SOAP-based Web service requests in a mobile environment can take 10 times longer and consume eight times more memory than an equivalent RESTful Web service request. The difference between these findings and ours might be explained by the fact that mobile devices are resource-constrained (and therefore require more processing time) compared to the laptops used in our experiments.

Related Work in Comparing SOAP- and REST-Based Applications

A few existing papers have contrasted SOAP-based Web services programmatic interfaces with their counterparts that conform to the REST architecture, and are thus "RESTful." In their paper, for example, Cesare Pautasso and colleagues provide a technical comparison of the two programmatic interfaces, including architectural principles, conceptual decisions, and technology decisions.[1] From an architectural principles standpoint, they compare 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. From a technical viewpoint, they compare transport protocols, payload format, service discovery, service composition, and so on. The paper offers several insights related to the theoretical comparison of the two interfaces; unfortunately, it provides little practical insight because it doesn't focus on concrete case studies that include prototypes and measurements.

Another example is work by Michael zur Muehlen and his colleagues.[2] Their comparison focuses on the choreography standards that come with the two interfaces; and they consider the standardization processes, the standard specifications, and public comments on the specifications. The study offers insights that are clearly of interest to standardization stakeholders. Unfortunately, it provides no practical information on the differences between the two interfaces in terms of concrete application development.

Several papers also deal with multimedia conferencing application development with SOAP-based Web services programmatic interfaces. However, they rarely attempt to contrast these applications with RESTful Web services counterparts. In their work, Miguel Gómez and Tomás P. de Miguel address the general issue of SOAP-based Web service multimedia conferencing in the IP Multimedia Subsystem.[3] IMS is the service delivery platform of next-generation telecommunications networks; Gómez and de Miguel discuss the two programmatic interfaces that IMS supports – namely, the Open Service Access conference management API and the SOAP-based Parlay-X conferencing interface that we use here. They also describe the technologies' shortcomings and argue that the most promising solution is to extend the SOAP-based Parlay-X conferencing interface.

Feng Liu and his colleagues deal with sessions-based applications at large in converged communication services settings,[4] but they don't focus on conferencing applications. They propose a SOAP-based Web service interface that lets a client in a session act as a SOAP end point, receiving and parsing SOAP messages sent from a centralized application. The same client can also act as a SIP end point. Wu Chou and his colleagues also deal with sessions-based applications at large, with no focus on conferencing applications.[5] They propose 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 using RESTful Web services for multimedia conferencing applications, but without comparing them to SOAP-based Web services counterparts. In their work, for example, David Lozano and his colleagues focus on IMS and aim at the convergence of telecommunications networks and Web 2.0.[6] They propose a reference model and sketch out an embryonic RESTful conferencing model. The model includes the state, list of participants, media description, and a link for media components as resources. However, the model hasn't been implemented, and the authors don't describe any concrete prototype. Several other papers have researched using RESTful Web services as programmatic interfaces in IMS, but for applications other than multimedia conferencing. One example focuses on instant messaging and presence service, and specifies the interfaces and describes the prototype.[7] Another example focuses on presence and describes a proof-of-concept prototype that uses an IMS application with avatars as mashup parts for blogs, SMSs, and schedule systems.[8] Yet another example deals with streaming services, focusing on user-generated video content and proposing a framework for video streaming over IMS.[9] A prototype, developed with a RESTful Web service interface, makes the framework available to developers.

References

1. C. Pautasso et al., "RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision," *Proc. 17th Int'l Conf. World Wide Web*, ACM, 2008; doi:10.1145/1367497.1367606.
2. M. Muehlen et al., "Developing Web Services Choreography Standards – The Case of REST vs. SOAP," *Elsevier Decision Support Systems*, vol. 40, 2005, pp. 9–29.
3. M. Gomez and T. de Miguel, "Advanced IMS Multipoint Conference Management Using Web Services," *IEEE Comm.*, vol. 45, no. 7, 2007, pp. 51–57.
4. F. Liu et al., "WSIP – Web Service SIP Endpoint for Converged Multimedia/Multimodal Communication Over IP," *IEEE Int'l Conf. Web Services*, IEEE Press, 2004, pp. 690–697.
5. W. Chou et al., "Web Services for Communications over IP," *IEEE Int'l Conf. Web Services*, IEEE Press, 2007, pp. 372–379.
6. D. Lozano et al., "WIMS 2.0: Converging IMS and Web 2.0 – Designing REST APIs for the Exposure of Session-Based IMS Capabilities," *Proc. 2nd Int'l Conf. Next-Generation Mobile Applications, Services and Technologies*, IEEE Press, 2008, pp. 18–24.
7. H.M Rissanen et al., "Design and Implementation of a RESTful IMS API," *Proc. 6th Int'l Conf. Wireless and Mobile Communications*, IEEE Press, 2010; doi:10.1109/ICWMC.2010.30.
8. N. Takaya et al., "Presence with Avatar for Web 2.0 – IMS Services Using REST Interfaces," *Proc. 12th Int'l Conf. Intelligence in Next Generation Networks*, IEEE Press, 2008; www.icin.biz/files/2008papers/Session4B-1.pdf.
9. D. Patnaik et al., "A Framework for Converged Video Services in the IP Multimedia Subsystem," *Proc. 3rd IEEE Int'l Conf. Internet Multimedia Services Architecture and Applications*, IEEE Press, 2009, pp. 95–100.

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 SOAP overhead in terms of network load. This overhead is due to the mandatory SOAP body (which, for instance, must be sent to get conference information, while the counterpart RESTful request's body is empty) and the extra information added in SOAP messages (such as the SOAP envelope). The network load overhead is two to three times less when using a REST interface.

Acknowledgments

The Canada Research Chair in End User Service Engineering for Communications Networks partially supported this work.

References

1. E. Newcomer, *Understanding Web Services: XML, WSDL, SOAP, and UDDI*, Addison-Wesley, 2002.
2. R.T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," doctoral dissertation, Dept. of Information and Computer Science, Univ. of California, Irvine, 2000.
3. *Parlay X Web Services; Part 12: Multimedia Conference*, 3GPP Global Initiative, 2010.
4. F. Belqasmi et al., "RESTful Web Services for Service Provisioning in Next Generation Networks: A Survey," *IEEE Comm.*, vol. 49, no. 12, 2011, pp. 66–73.
5. J. Rosenberg et al., *SIP: Session Initiation Protocol*, IETF RFC 3261, June 2002; www.ietf.org/rfc/rfc3261.txt.
6. J. Van Dyke et al., *Media Server Control Markup Language (MSCML) and Protocol*, IETF RFC 5022, Nov. 2007; http://tools.ietf.org/html/rfc5022.
7. M. Hadley and P. Sandoz, *JAX-RS: Java API for RESTful Web Services*, *Version 1.1*, Sun Microsystems, Sept. 2009.
8. M. Kulkarni et al., *SIP Servlet Specification, version 1.1*, JSR 289 Expert Group, Aug. 2008.
9. F. AlShahwan and K. Moessner, "Providing SOAP Web Services and RESTful Web Services from Mobile Hosts," *Proc. 5th Int'l Conf. Internet and Web Applications and Services* (ICIW), IEEE Press, 2010, pp. 174–179.

Fatna Belqasmi is a research associate at Concordia University, Canada. Her research interests include next-generation networks, service engineering, distributed systems, and networking technologies for emerging economies. Belqasmi has a PhD in electrical and computer engineering from Concordia University, Canada. Contact her at fbelqasmi@alumni.concordia.ca.

Jagdeep Singh is a graduate student in electrical and computer engineering at Concordia University. His research interests include Web services, services for next-generation networks, and the IP Multimedia Subsystem (IMS). Singh has a BEng in electronics and communication engineering from Thapar University, India. Contact him at jagd_si@encs.concordia.ca.

Suhib Younis Bani Melhem is a doctoral student in electrical and computer engineering at Concordia University. His research interests include RESTful Web services, multimedia conferencing, and 4G networks. Melhem has an MEng in electrical and computer engineering from Concordia University and an MEng in computer engineering from Jordan University of Science and Technology. Contact him at s_banim@encs.concordia.ca.

Roch H. Glitho 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. His research interests include architectures for end-user services, distributed systems, nonconventional networking, and networking technologies for emerging economies. Glitho has a PhD in tele-informatics from the Royal Institute of Technology, Stockholm, Sweden. Contact him at glitho@ece.concordia.ca; www.ece.concordia.ca/~glitho.