Received November 24, 2017, accepted January 4, 2018, date of publication January 15, 2018, date of current version March 13, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2794258

A Cloud-Based Architecture for Multimedia Conferencing Service Provisioning

ABBAS SOLTANIAN[1], FATNA BELQASMI[2], SAMI YANGUI[1,3], MOHAMMAD A. SALAHUDDIN[4], (Member, IEEE), ROCH GLITHO[1,5], AND HALIMA ELBIAZE[6]
[1]Concordia University, Montreal, QC H3G 1M8, Canada
[2]Zayed University, Abu Dhabi 144534, United Arab Emirates
[3]LAAS-CNRS, Université de Toulouse, INSA, 31400 Toulouse, France
[4]David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada
[5]University of Western Cape, Cape Town 7535, South Africa
[6]Université du Québec à Montréal, Montreal, QC H2L 2C4, Canada

Corresponding author: Abbas Soltanian (ab_solta@encs.concordia.ca)

This work was supported in part by the NSERC SAVI Research Network, in part by the NSERC Discovery Grant Program, in part by the FQRNT New Researcher Start-Up Program, and in part by the Canada Research Chair in End-User Service Engineering for Communications Networks.

ABSTRACT Multimedia conferencing is the real-time exchange of multimedia content between multiple parties. It is the basis of several interactive multiuser applications, such as distance learning and multimedia multiplayer online games. The cloud-based provisioning of the conferencing services on which these applications rely on can have several benefits, including the easy provisioning of new applications, efficient use of resources, and elastic scalability. This paper proposes a holistic cloud-based architecture for conferencing service provisioning, which covers both the infrastructure and platform layers of the cloud. The proposed infrastructure layer offers conferencing substrates-as-a-service (e.g., dial-in signaling, video mixing, and audio mixing), instead of virtual machines or containers. The platform layer abstracts the details of the conferencing concepts and offers a high-level interface to simplify conference service provisioning for a wide range of service and application providers (experts versus non-experts). It also enables the on-the-fly scaling of the running conferences while guaranteeing the required quality of service, enables substrates composition to create new conferencing services, and eases the reuse of conferencing services in building new applications. The presented architecture is supported by a proof-of-concept prototype and performance measurements. The latter provides the analysis of resource allocation efficiency and response time, as well as the scalability of the system under suboptimal and over-provisioned conditions. It also provides recommendations for service providers regarding the best alternatives for provisioning their service.

INDEX TERMS Cloud computing, conferencing service provisioning, infrastructure-as-a-service, multimedia conferencing, platform-as-a-service, substrate-as-a-service.

I. INTRODUCTION

Cloud computing is a paradigm for swiftly provisioning a shared pool of configurable resources (e.g., storage, network, applications, and services) on demand. It has three key facets: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS) [1]. It has several benefits, such as the rapid provisioning of new services, scalability, and elasticity. Multimedia conferencing is the conversational exchange of media content (e.g., voice, video and text) between multiple parties [2]. It is an important component of *conferencing applications* (e.g., online video meeting, distance learning, and massively multiplayer online games).

For cost efficiency and to cut the development time, conferencing application providers can use *conferencing services* (e.g., dial-in video conferencing and dial-out audio conferencing) offered by third parties. Such services could be provisioned as SaaS using a PaaS that eases their development. Conferencing services can themselves rely on basic conferencing building blocks (e.g., signaling, audio mixer and video mixer), referred to in this paper as the conferencing substrates. *Conferencing service provisioning* refers to the entire life-cycle of the conferencing service, i.e., development, deployment, and management [3]. Provisioning conferencing services in the cloud is quite challenging. One challenge for the conferencing service providers, for instance, is to master low-level details of conferencing technologies, protocols and their dependencies. Another challenge is scaling the size of the provisioned conferences with respect to the number of participants. The size of a conference needs to scale on-the-fly to accommodate a fluctuating number of participants while the conference quality of service requirements (e.g., latency) are met. The existing PaaS and IaaS solutions do not address these challenges.

This paper proposes a holistic conferencing cloud architecture to tackle those challenges. The new architecture provides novel application programming interfaces (APIs) to simplify the provisioning of the conferencing services for a wide range of service providers (experts vs. non-experts) and describes the process of substrate-as-a-service (SubaaS) composition. The SubaaSs are provided as RESTFul web services, a design style that reuses Web technologies. The architecture also allows the application providers to utilize the offered conferencing services without having to deal with the complexities of conferences. For scalability, the architecture relies on cloud computing. It covers both the PaaS and IaaS layers and it is an extension to our previous cloud-based conferencing architecture that only covers PaaS [4].

The architecture is based on the business model in [2], which introduces six roles: connectivity provider, broker, conferencing substrate provider, conferencing infrastructure provider, conferencing platform provider and conferencing service provider. This paper reuses and extends this business model by adding a new role, entitled the conferencing application provider. It also assumes that the conferencing infrastructure provider plays the role of the substrate provider too. In this architecture, the infrastructure provider exposes the conferencing substrates as services (SubaaS) to the platform (i.e., PaaS) provider. The PaaS provider offers high-level APIs to create innovative conferencing services and it enables the on-the-fly composition of SubaaS into full-fledged conference services. The conferencing application providers reuse the conferencing services offered as SaaS in building new applications. They also use PaaS to update the running conferences in their applications at runtime (e.g., switching from audio conference to audio/video conference) without stopping the ongoing conferences.

The rest of the paper is organized as follows: Section II gives required background information. Section III introduces a motivating scenario, derives requirements, and reviews related works. Section IV describes the proposed overall architecture. Section V presents the implementation architecture, prototype, and experimental results. Section VI concludes the paper.

II. BACKGROUND INFORMATION

This section gives background information on multimedia conferencing, cloud platforms and infrastructures, and RESTful service composition.

A. MULTIMEDIA CONFERENCING

Multimedia conferencing has three main architectural components, namely signaling, media handling, and conference control [2]. Signaling is in charge of the establishment, modification and tear down of multimedia sessions. Session establishment can be done in three different ways: dial-in, dial-out and ad-hoc. In dial-in conferences, the participants should call the signaling server to join the conference while in dial-out conferences, the server calls all the participants. An ad-hoc conference starts when two participants are in a call and more participants are invited to join, by those already in the conference. Media handling is about media functionalities such as audio mixing, video mixing, and transcoding. Conference control encompasses control functions such as floor control, allowing the management of shared resources (e.g., audio channel) in the conference.

B. CLOUD PLATFORMS

Many cloud platforms are available for the development and management of the applications offered as SaaS. Examples of the existing PaaSs are Google Cloud Platform, Aneka, and Cloud Foundry. There are also some reference models for the PaaS architecture, such as the one introduced by IBM [5]. This PaaS layered architecture consists of four layers: (1) *Front-end* – with a set of user and developer APIs and tools. Development APIs allow developers to allocate and manage the PaaS resources. The user APIs and GUIs allow the end-users to invoke and execute the running applications in the PaaS. (2) *Core* – with necessary frameworks (e.g., containers and storage services) required for application hosting and execution. (3) *Management and Governance* – consisting of entities for managing the PaaS and the hosted applications (e.g., monitoring, scalability). Moreover, it has the required entities to support the PaaS business model (e.g., billing, membership). (4) *Abstraction Interface* – with a set of APIs that enable interactions with the underlying IaaS.

C. CLOUD INFRASTRUCTURE

There are several existing cloud IaaS providers today, such as Amazon EC2 and Microsoft Azure. Similar to the PaaS architecture, IaaS also has some reference architectures such as the one introduced in [6]. It consists of three main layers: (1) *Cloud Management* – in charge of managing the overall IaaS. It also acts as an interface with IaaS consumers (e.g., PaaS, another IaaS, and the Clouds). (2) *Virtual Infrastructure Management* – providing uniform and homogenous view of virtual resources. It provides primitives to schedule and manage VMs across multiple physical hosts. (3) *Virtual Machine Management* – providing simple primitives (e.g., start, stop, suspend) to manage VMs on a single host.

D. RESTful SERVICE COMPOSITION

Representational State Transfer (REST) is an architectural style based on the resources associated with unique identifiers (e.g., URI) [7]. The interactions with these resources are based on a standardized communication protocol (e.g., HTTP) and operations (i.e., GET, POST, PUT and DELETE). The web services that are implemented following REST guidelines are referred to as RESTful services.

There are different approaches to RESTful service composition, including orchestration and choreography [8]. Based on the World Wide Web Consortium (W3C) definition, the choreography defines the sequences and conditions where different independent services exchange data while orchestration defines the sequences and conditions where one service invokes other services [9]. In other words, orchestration allows a central entity to control different services and their interactions while choreography allows the individual services to collaborate in a decentralized manner.

Service selection is critical for service composition. For instance, to create a new conferencing application, adequate conferencing services (i.e., offered as SaaSs) should be selected with the appropriate characteristics. The same applies for creating a conferencing service from the existing conferencing substrates (i.e., offered by the conferencing IaaS). Various selection algorithms exist. Examples are presented in [10] and [11]. These algorithms decide about the costs and the gains, which can refer to different parameters such as price and response time. Based on the Service Oriented Architecture (SOA) [12], each service provider can publish the information of the services in a service broker. The service broker acts as a directory for the published services. Then, the service selection algorithm can discover the suitable services from the broker.

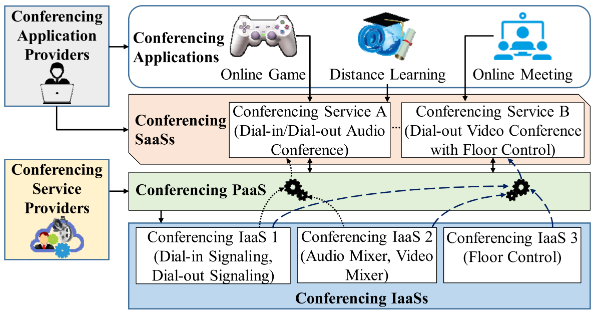

Conferencing Application Providers
Conferencing Applications
Online Game
Distance Learning
Online Meeting
Conferencing SaaSs
Conferencing Service A (Dial-in/Dial-out Audio Conference)
Conferencing Service B (Dial-out Video Conference with Floor Control)
Conferencing Service Providers
Conferencing PaaS
Conferencing IaaS 1 (Dial-in Signaling, Dial-out Signaling)
Conferencing IaaS 2 (Audio Mixer, Video Mixer)
Conferencing IaaS 3 (Floor Control)
Conferencing IaaSs

FIGURE 1. **Scenario for conferencing service provisioning in the cloud.**

III. MOTIVATION, REQUIREMENTS AND RELATED WORK

A. MOTIVATING SCENARIO

Fig. 1 depicts the motivating scenario. There are conferencing application providers that use conferencing services offered as SaaS, to develop their applications. Three conferencing applications are provisioned: (1) an online game that allows dial-in audio conferencing between the game players, (2) a distance learning application that enables dial-out audio conferencing between students and teachers, and (3) an online meeting application that offers dial-out video conferencing with floor control. The conferencing service providers in the scenario use the conferencing PaaS to provision the conferencing services these applications are based on. One service provider offers Conferencing Service "A" that supports both dial-in and dial-out audio conferences. The distance learning and the game applications utilize Service A. Another conferencing service provider offers a dial-out video conference service with floor control, i.e., Service B. This second service is used by the online meeting application.

The conferencing SaaSs create new conferences when they receive corresponding requests from the conferencing applications. For example, Service A creates a dial-in audio conference when it receives a request from the game application. To run the conference, PaaS finds the appropriate SubaaSs (i.e., dial-in signaling and audio mixer in this example), composes them, and requests the relevant IaaS(s) to create and activate an adequate instance of each substrate (e.g., the audio mixer with the capability of supporting 500 users). The SubaaSs involved in a given composed conference application may belong to different substrate/IaaS providers. As the players join and leave a conference, PaaS scales the conference up and down in terms of the number of participants. Then, the conferencing IaaS should scale the corresponding instances up and down in terms of the virtualized hardware (e.g., CPU, RAM, and Storage) and software (e.g., the number of running instances of each substrate). Scaling in both layers is done in an elastic manner.

B. REQUIREMENTS

The following five sets of requirements are derived from the motivating scenario:

1) HIGH-LEVEL NORTHBOUND PAAS INTERFACES FOR SERVICE PROVIDERS

The conferencing PaaS northbound interfaces should enable the service providers to provision new services without having to deal with the complexities of conferencing components and their interactions. The interfaces should also be flexible enough for creating complex and novel conferencing services (e.g., a dial-in audio conference with five minutes of chat per hour).

2) COMPOSITION OF CONFERENCE SERVICES FROM SubaaSs

This feature enables and simplifies creating complex conferencing services based on the basic substrates that are offered as services. For example, the dial-in audio conference service (Conferencing Service A in the scenario) is composed of a dial-in and audio mixer SubaaSs.

3) ELASTIC SCALABILITY

The conferencing PaaS, in collaboration with the conferencing IaaSs, should be scalable in terms of different conferencing concepts such as the number of conferences, sub-conferences, floors, and conference participants. Scaling in an elastic manner allows the pay-as-you-go [13] principle of the Cloud.

4) MEETING QUALITY OF SERVICE

Meeting the Quality of Service (QoS) requirements, such as latency, jitter, and throughput, is critical as conferencing services are real-time.

5) PUBLISH-AND-DISCOVERY MECHANISM

This feature allows the conferencing application providers to find the appropriate conferencing services that can fulfill their requirements. It also enables the conferencing PaaS to discover a conferencing IaaS and a conferencing IaaS, to discover other conferencing IaaSs for excess workload distribution.

C. RELATED WORK

The cloud-based conferencing architectures and the existing PaaS and IaaS related solutions are reviewed below. Service composition and discovery solutions are also discussed.

1) CLOUD-BASED CONFERENCING ARCHITECTURES

The existing architectures can be categorized with focus on the SaaS or IaaS layers. Examples of the first category are presented in [12] and [14]. The two solutions focus on developing cloud conferencing services at the application layer, without addressing the challenges related to the PaaS and IaaS layers (e.g., scalability, QoS, publication, and discovery of conferencing services). Reference [12] offers conferencing services as SaaS, while using a conventional PaaS for deployment and execution. No high-level interfaces are provided to ease the development and management of such services. Reference [14] presents an approach for providing video conferencing as a web service, defines the interfaces to be used by the conferencing application providers. This work tries to transform the existing telecommunication services into a reusable resource for the third parties. However, it does not address how this service is provisioned.

Reference [15] is an example of relevant works with a focus on the IaaS layer. The proposed architecture relies on conferencing substrates and enables elastic scalability. It also proposes PaaS/IaaS interfaces rooted in substrates and proposes a broker between IaaS and PaaS that allows finding suitable substrates. However, it does not consider the PaaS and SaaS layers and their relevant issues. Neither does it include high-level PaaS interfaces for service providers, address substrates' composition or provide a support for QoS.

Other works in the relevant literature, such as [16]–[18], address specific problems of cloud-based conferencing, such as inter-datacenter network utilization, media mixing, and transcoding. While they focus on how conferencing components can efficiently utilize the cloud, they do not address conferencing service provisioning. In addition, as these works only offer one service, they do not tackle the service publication, discovery, and composition.

2) EXISTING PaaS SOLUTIONS

Aneka [19] and Cloud Foundry [20], the two PaaS representatives, are evaluated. Aneka provides high-level interfaces and supports elastic scalability, specifically for distributed application provisioning. Nonetheless, it does not offer any conferencing APIs. Cloud Foundry provides no interfaces for conferencing service provisioning. It supports the scaling of application instances but does not address scaling in terms of conference concepts. Neither does it address composition and QoS.

3) EXISTING IaaS SOLUTIONS

Some relevant literature propose a conceptual architecture of open-source IaaSs. Reference [21], for example, proposes the OpenStack architecture that consists of five layers: Compute (Nova), Storage (Swift), Image (Glance), Identity (Keystone) and Dashboard (Horizon). Nova is the computing fabric controller for OpenStack and it is all about access to the computing resources. Swift, as the storage infrastructure in OpenStack, offers APIs to store and retrieve lots of data. Glance builds a discovery and retrieval system for VM images. Keystone is responsible for authentication and authorization. Horizon provides a web-based user interface to all above OpenStack services. In [22], instead of having one layer for Storage, it is broken down into two layers: Block Storage and Object Storage. Block Storage offers storage volume for Compute layer while Object Storage stores the actual virtual disc files. Their architecture also has a Network layer to provide virtual networking for the Compute layer. All components in both architectures are following a shared-nothing policy, meaning each component can be installed on any server.

The OpenNebula architecture proposed in [21] and [23] has three layers: Drivers, Core, and Tools. Drivers do the communication with the underlying operating system. VM creation, startup and shutting down are parts of this layer's functionality. The core is a centralized layer that manages the VM life cycle. To manage VMs, Tools offers different interfaces for communication with users. Sotomayor *et al.* [6] keep the Core and Drivers layers and propose Scheduler to replace Tools. Scheduler decides about VM placement. This layer keeps track of all the incoming requests in order to send an appropriate deployment command to the Core layer, based on those requests. They also have an Interface layer to communicate with users.

All above IaaS solutions are VM-based, thus, their interfaces should change to support the communication rooted in conferencing concepts (e.g., start, stop and modify the conferencing substrates). Moreover, they support scalability in terms of computing resources, storage, and networking. However, as a conferencing IaaS, there is a need to scale resources in terms of conferencing concepts (e.g., the number of participants) to collaborate with the conferencing PaaS.

4) SERVICE COMPOSITION AND DISCOVERY

RESTful web service composition is a well-researched topic as several solutions and alternatives have been proposed to cater to different situations [24]–[26].

Service composition can be done in a static or dynamic way [8]. In a static composition, the basic services as part of the composition are selected in advance and their aggregation takes place at the design time. In contrast, dynamic composition allows to select and replace the basic services during the runtime. The composition can also be done manually, semi-automated or automatically [8]. In manual composition, the service provider should define and create an abstract composite process and manually bind the services to the abstract process. Some web service standard languages such as BPEL [27] or OWL-S [28] can be used to create the abstract process. In automatic composition, the new composite service specification can be generated by selecting adequate services based on the specified requirements [8]. Semi-automatic composition leverages both manual and automatic approaches. Workflow-based and template-based compositions are other composition planning techniques [29]. In the workflow-based composition, the process is depicted as an acyclic directed graph with control and dataflow. This technique requires the developers' extensive domain knowledge and is time-consuming. In the template-based composition, templates describe the outline of activities required to solve the problem. Templates are parameterized and use variables that allow customization based on the users' needs and preferences. In fact, the templates lead to creating an executable workflow.

Reference [30] proposes a cloud service broker to facilitate the deployment of Cloud application topologies from multiple Cloud providers. The authors also propose a multi-criteria optimization algorithm to select the basic services to be composed. The algorithm sets cost efficiency as the main objective. Yangui *et al.* [31] consider a wide range of objectives to design their cloud broker selection mechanism, such as user constraints, financial, energetic, geographic or operator contractual preferences. Reference [32] considers multimedia conferencing requirements for designing the service broker. The authors here propose an architecture for substrate service publication and discovery. Their service broker acts between the substrate providers and the conferencing IaaSs and offers some REST APIs as the interfaces between them.

IV. PROPOSED CONFERENCING ARCHITECTURE

In this section, the architectural principles are presented. Then, the architectural components and service development APIs are discussed in detail, followed by an illustrative scenario.

A. ARCHITECTURAL PRINCIPLES

The first principle is to adopt the orchestration approach for the SubaaS composition because it provides PaaS with a greater control on the substrates and their interactions.

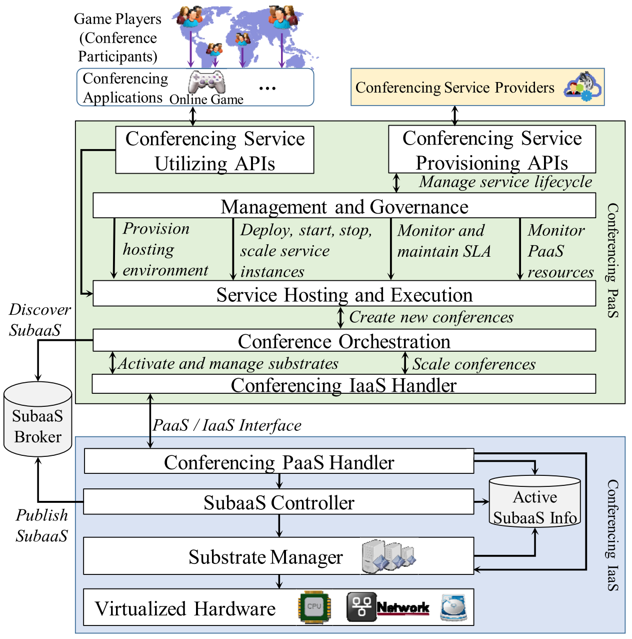

Game Players (Conference Participants)
Conferencing Applications
Online Game
...
Conferencing Service Providers
Conferencing Service Utilizing APIs
Conferencing Service Provisioning APIs
Manage service lifecycle
Management and Governance
Provision hosting environment
Deploy, start, stop, scale service instances
Monitor and maintain SLA
Monitor PaaS resources
Service Hosting and Execution
Create new conferences
Conference Orchestration
Activate and manage substrates
Scale conferences
Conferencing IaaS Handler
Conferencing PaaS
Discover SubaaS
SubaaS Broker
PaaS / IaaS Interface
Publish SubaaS
Conferencing PaaS Handler
SubaaS Controller
Active SubaaS Info
Substrate Manager
Virtualized Hardware
CPU
Network
Conferencing IaaS

FIGURE 2. Overall cloud-based conferencing architecture.

The second principle is to use high-level PaaS/IaaS interfaces rooted in the conferencing substrates. This principle enables PaaS to request IaaSs for scaling conferences in terms of conference concepts (e.g., the number of participants) rather than VM or the container resources. The third principle is to leverage the existing PaaSs and IaaSs. This allows reusing the existing solutions for the conferencing PaaS and IaaS implementation. The last principle is that the conferencing IaaSs expose substrates as RESTful web services. Therefore, existing approaches and techniques for RESTful web service orchestration, such as [33], can be reused in the conferencing PaaS for substrates composition.

B. GENERAL ARCHITECTURE

The proposed cloud-based conferencing architecture, as shown in Fig. 2, includes two main layers (i.e., PaaS and IaaSs) and a broker. The figure also shows the conferencing service providers, the conferencing applications, and the conferencing application users referred to as the conference participants. Note that PaaS may need to communicate with multiple IaaSs to provision a given conferencing service.

1) PaaS COMPONENTS

The PaaS layer consists of six components, which deal with two key facets: (a) conferencing service provisioning and utilizing, and (b) conference management.

a: CONFERENCING SERVICES PROVISIONING AND UTILIZING

This facet covers conferencing SaaSs development, deployment, and management in addition to conferencing SaaSs utilizing. It includes four components. The *Conferencing Service Provisioning APIs* component offers high-level APIs to the conferencing service providers, for easy provisioning of new conferencing SaaSs. It also allows the SaaSs providers to make their services available to the application developers via publishing them into a PaaS local service repository.

The *Conferencing Service Utilizing APIs* provides high-level APIs for conferencing application providers, to discover (from the local service repository), reuse, and control the existing conferencing SaaSs.

The *Management and Governance* component manages the conferencing services and monitors their QoS and SLAs during service execution. It deploys and executes new services in the *Service Hosting and Execution* component, upon receiving the requests from the *conferencing Service Provisioning APIs*.

The *Service Hosting and Execution* component hosts the conferencing services. It allocates necessary PaaS resources (e.g., server runtime and database drivers) and prepares the execution environment before hosting.

Note that the *Conferencing Service Provisioning* and *Utilizing APIs* are the extensions of the application provisioning front-end available in regular PaaS architectures. The *Management and Governance*, as well as the Service Hosting and Execution components are reused from the conventional PaaS architectures.

b: CONFERENCE MANAGEMENT

This facet concerns the management of the actual conferences (i.e., the virtual rooms where people can meet and communicate). It encompasses conference creation as well as the management of the created conferences (e.g., scaling the size of a conference to support more participants). The main component of this facet is *Conference Orchestration* with the following five tasks: First, it determines the necessary substrate types and their associated requirements by using, for instance, syntactic matching with the categorized API parameters. This task starts upon receiving the execution or modification request for a specific conferencing SaaS. Second, based on the determined types and requirements, it discovers the most suitable conferencing SubaaSs from the broker. The existing algorithms for cloud service selection, such as [10], can be reused in this context. Third, it orchestrates conferences from the selected SubaaSs and executes them. Note that conferences are executed in this component. In contrast, the conferencing SaaSs that create conferences are executed in the Service Hosting and Execution component. Fourth, it manages the composed conferences. For example, it can add the video mixing ability to a conference or remove it from it. Fifth, it monitors the running conferences to make decisions if any scaling is required. For instance, if the number of participants in a conference increases, it decides to scale the conference size. Thus, it requests the conferencing IaaSs to scale the corresponding substrates to cope with the new workloads.

Another component under this facet is*the Conferencing IaaS Handler*, which is in charge of communications between the conferencing PaaS and the conferencing IaaSs. For instance, a scaling request initiated by the *Conference Orchestration* component is sent to the corresponding conferencing IaaSs through the *Conferencing IaaS Handler*. Note that *Conference Orchestration* is a novel component while *Conferencing IaaS Handler* is an extension of IaaS communication component in conventional PaaS architectures.

2) IaaS COMPONENTS

The IaaS layer consists of five components, dealing with two key facets: (a) resource management and (b) SubaaS management.

a: RESOURCE MANAGEMENT

This facet is in charge of providing required resources in order to run a substrate. The *Virtualized Hardware* is one of the components in this facet. It has a pool of typical virtualized IaaS resources such as CPU, Network, and Storage. The second component of this facet is *Substrate Manager* with three main tasks: First, it creates and hosts resources in order to run the substrates. These resources can be a VM or a container [34] that uses virtualized hardware to host a substrate. Each substrate can be hosted on one or many VMs or containers (e.g., two instances of the same substrate may be activated in two different machines). In addition, each VM or container may host more than one substrate. The second task is modifying the allocated resources upon receiving the scaling request for a substrate. For instance, to scale up a running substrate, it can add some virtualized hardware to the VM that hosts the target substrate. The third task is inserting and updating the information of all running substrates in a repository called *Active SubaaS Info*.

b: SubaaS MANAGEMENT

This facet includes the managing functionalities to offer substrates as services. The first component of this facet is the *Active SubaaS Info*. It is a repository that keeps information about all running SubaaSs. For instance, for each running SubaaS, it keeps the related conference ID, IP of the VM(s) or container(s) hosting that substrate, etc.

Another component of this facet is *SubaaS Controller*. This component has two main tasks. First, it decides how and when to scale a running substrate, based on the Service Level Agreements (SLAs) between the PaaS and IaaS (e.g., end-to-end delay should be less than 400 msec). Upon receiving the scaling request from the PaaS and its required QoS, it uses the stored information in the Active SubaaS Info repository to make the scaling decisions. The resource allocation algorithm and video mixing procedure we proposed earlier in [35] is used for this purpose. Second, it maintains a repository of all available substrates in the IaaS. It selects the suitable substrate from this repository when it receives a request to create and start a substrate. It then instructs the Substrate Manager to create the actual resource. Moreover, it publishes the information of SubaaSs in the broker.

The third component of this facet is *Conferencing PaaS Handler*, which is in charge of all communications between

TABLE 1. Examples of conferencing service development APIs.

REST Resource	Operation	HTTP action and resource URI	Request body parameters	Most important info in response
List of Conferences	Create conference	POST: /conferences	Conference model, Media, Conference technology, floor control, conference size, QoS requirements	ID and URI of the created conference resource
List of participants	Add participant	POST: /conference/ {conferenceId}/ participants	Participant description: name, URI	ID and URI of the new participant resource
List of floors	Add floor	POST: /conferences /{conferenceId} /floors	Floor description: chair, floor participants	ID and URI of the newly created floor resource
Specific sub-conference	Remove subconference	DELETE: /conferences /{conferenceId} /subconferences /{subconferenceId}	None	Success or failure indication

the PaaS and IaaS layers. This component has two main tasks: First, it receives and dispatches the PaaS requests (e.g., to create a substrate and scale up a substrate) to the appropriate IaaS components and forwards the IaaS replies to the PaaS. Second, it handles the conference participants' requests (e.g., joining a conference). The participants' requests are sent from the conferencing applications to the PaaS, which forwards them to the conferencing IaaS. The *Conferencing PaaS Handler*, in collaboration with the Active SubaaS Info repository, identifies the appropriate substrates and forwards the requests to them. This feature increases the level of abstraction for the substrates working in a single conference. Moreover, there is no need to update the participants on any changes in the substrates' hosting resources.

3) BROKER

The Broker lists the SubaaSs offered by different IaaSs. The SubaaSs description is semantic-based to allow for rich descriptions and queries. It includes high-level information such as the type of service, QoS parameters, and cost. In this paper, we reuse the description model and the broker publication and discovery interfaces from [32].

C. CONFERENCING SERVICE DEVELOPMENT APIs

Three principles are followed to design the proposed APIs. The first principle is leveraging basic conferencing concepts (e.g., conference, participant, media, and floor) in the API design. This helps in achieving an abstraction level higher than conferencing components (e.g., signaling, media mixer and media transcoder) and their complex interactions. The second principle is categorizing API parameters, which helps service providers to easily understand conference mandatory and optional aspects, required API parameters for each aspect and dependencies among parameters. The third principle is the use of RESTful design. It is standard-based, lightweight and flexible for data representation, which allows describing the APIs in a generic way.

Table 1 delineates four API examples. It shows some of the REST resources along with an example operation for each. The request parameters and the response contents are also listed. Showing the categorization of API parameters, table 2 highlights that a service provider has to specify one conference model, at least one media and the conferencing technology. It also shows the conditional dependencies of parameters. For example, for WebRTC-based conferencing [36], signaling protocol must be specified. In this table, the parameters that the service providers can change during the runtime are italicized.

D. SERVICE COMPOSITION

As per our first design principle, the conferencing services are composed of SubaaSs using the orchestration approach. The *Conference Orchestration* component of the PaaS plays the role of the central entity that invokes and controls the composing SubaaSs.

In addition to the composition approach, two other composition aspects are considered: binding dynamicity and automation level [8]. Since the PaaS discovers, selects, and activates the composing SubaaSs on the fly, dynamic binding to IaaSs (i.e., SubaaS providers) is required. As for the automation level, the semi-automated approach is adopted to take advantage of more mature and widely used techniques, such as workflow.

In this work, the conferencing PaaS provider develops a generic workflow template for the composite conference, considering the various substrate types that may be required. It uses a workflow automation tool (e.g., Activiti [37]) to ease and speed up the process. When the *Conference Orchestration* component selects the SubaaSs to be composed (i.e., at runtime), it creates an instance of the workflow template and then configures the instance with the selected and activated substrate instances. Thus, the conference is dynamically bound to its composing substrate services. This dynamic binding makes it possible and easy to change the substrates used by an ongoing conference at runtime if needed. Note that a PaaS provider may define multiple workflow templates and then select the most suitable one based on the required substrate types and the rest of the users' requirements.

E. ILLUSTRATIVE SCENARIO

The illustrative scenario consists of (i) an online game application where players can talk for unlimited time but can have private text chat for only 5 minutes per hour, (ii) a service provider that offers dial-in audio conferencing as SaaS with text chat for a limited time and (iii) a conferencing PaaS that subscribes to three conferencing IaaSs: A, B and C, which offer dial-in signaling, audio mixing and instant

TABLE 2. Categorization of API parameters.

	Categories of Parameters	Example Values		
Mandatory Aspects	Conference Model	Pre-arranged conference	Dial-in conference	
			Dial-out conference	
		Ad-hoc conference		
	Media	At least one of audio, video, and text		
	Conferencing Technology	SIP-based	Signaling protocol	Default protocol: SIP. No need to specify
			Audio encodings	Default encoding: NULL. It should be specified
			Video encodings	Default encoding: NULL. It should be specified
		WebRTC-based	Signaling protocol	Default protocol: NULL. It should be specified
			Audio encodings	Default encoding: G.711 and Opus. Can specify additional
			Video encodings	Default encoding: H.264 and VP8. Can specify additional
		Hybrid (SIP + WebRTC based)	Mandatory protocols and encodings from both technologies apply. Can specify additional	
Optional Aspects	*Floor control*	At least one floor control policy, e.g. chair-moderated or round-robin		
	Subconference	Enabled or not		

messaging SubaaSs respectively. The scenario illustrates how the conferencing PaaS creates a conference when the game application sends a request to the conferencing SaaS and how the conferencing IaaSs allocate the resources.

Fig. 3 shows the interactions. For brevity, the game application is omitted in the figure. Using the Conferencing Service Utilizing APIs, the game application developer finds the offered conferencing services and requests for conferencing SaaS A. When conferencing SaaS A receives the game application request for creating a conference, it invokes the *create conference* API (step 1). The API handling is delegated to the *Conference Orchestration* component, which determines necessary substrate types (step 2) and finds appropriate SubaaSs through the broker (step 3). In this scenario, the dial-in signaling and the audio mixing SubaaSs are selected from IaaSs A and B respectively (step 4).

Next, the PaaS requests the IaaSs, via the *Conferencing IaaS Handler*, to activate the substrates (steps 5 to 12). For activation, the *Conferencing PaaS Handler* component in the IaaS receives the request and forwards it to the *SubaaS Controller.* The latter selects the requested substrate's code from its repository and sends the required information to the *Substrate Resource Manager* to allocate the required resources (e.g., it selects the audio mixer code that can handle 200 participants and asks the *Substrate Resource Manger* to create and run a new VM to accommodate 200 participants, install the substrate code on the VM, and run the code to initialize and activate the audio mixer as a substrate).

After activating the substrates, the *Conference Orchestration* binds the SubaaSs in the composing template (selected in step 2) and then executes the new dial-in audio conference (step 13). The orchestrated conference represents a full-fledged conference. Finally, the ID of the full-fledged conference is returned to the game (step 14).

It is assumed that the conferencing service enables private text chat after 30 minutes. When the timer expires, the service invokes the *addMedia* API to add instant messaging to the conference for 5 minutes (step 15). Thus, the *Conference Orchestration* discovers the appropriate SubaaS from the broker (step 16). It selects IaaS C, activates the instant messaging substrate and modifies the conference workflow to add instant messaging (step 17 to 22). On the new substrate, an individual conference is created for 5 minutes and the existing participants are added to it (step 23 to 26). A notification is sent to the game application (step 27) and the participants can start exchanging text messages. For optimization purposes, the messaging SubaaS can be added to the conference when created and it can be enabled and disabled when needed. Meanwhile, the messaging SaaS can be discovered and added at runtime if, for instance, the original one is no more available. The scenario is showing the latter case.

V. IMPLEMENTATION AND MEASUREMENTS

An implementation architecture is first presented. Next, the developed prototype is described and its validation and performance measurements are discussed.

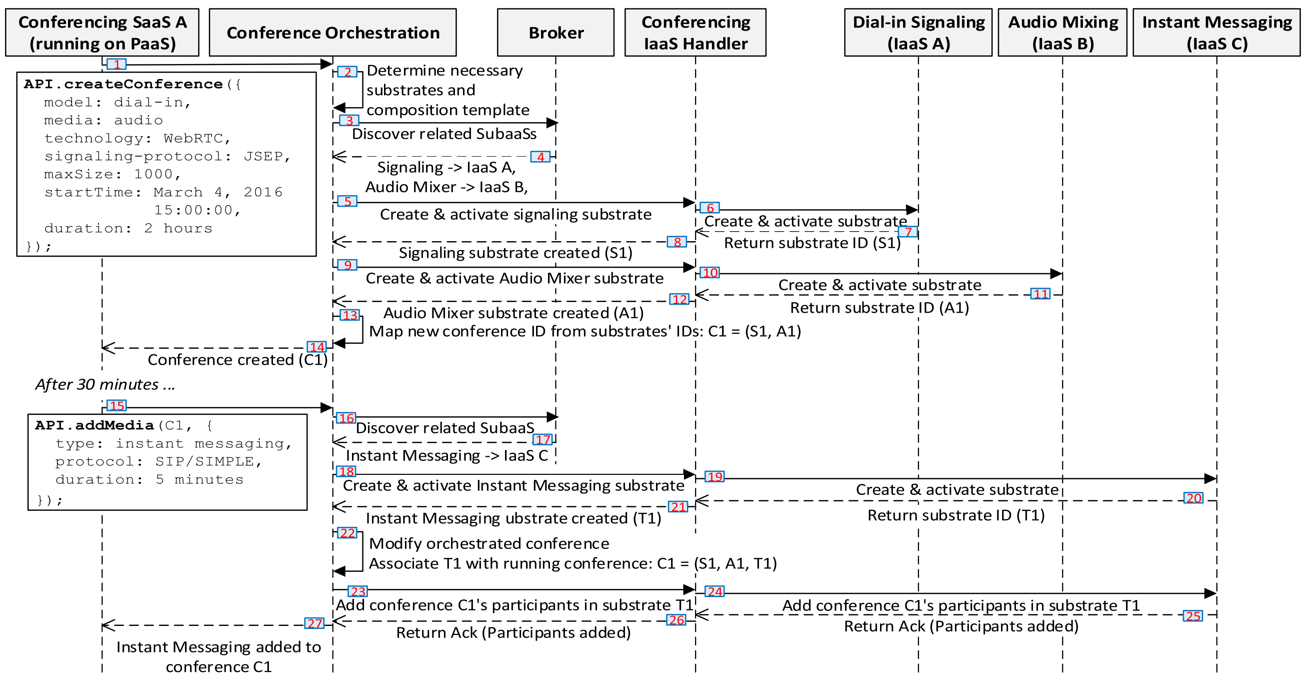
A. IMPLEMENTATION ARCHITECTURE

Fig. 4 shows the implementation architecture including Conferencing PaaS, Conferencing IaaS, and the SubaaS Broker.

1) CONFERENCING PaaS

In the Conferencing Service Provisioning APIs component, two sets of REST APIs are developed: *Conferencing SaaS Development APIs* and *Conferencing SaaS Deployment APIs*. These are used for service creation and deployment respectively. The Conferencing Service Utilizing APIs have been also implemented as REST APIs. Management and Governance and Service Hosting and Execution components are not discussed here as they are reused from the conventional PaaS architectures.

The *Conference Orchestration* component uses a repository to store the workflows of composing templates. The *Conference Manager* in this component receives the northbound requests for running conferences, selects an appropriate template from the repository, and determines the required substrates for the conference. It then sends that information to the *SubaaS Selector* and the *Substrate Orchestration Engine.*



```
API.createConference({
  model: dial-in,
  media: audio
  technology: WebRTC,
  signaling-protocol: JSEP,
  maxSize: 1000,
  startTime: March 4, 2016
             15:00:00,
  duration: 2 hours
});
```

```
API.addMedia(C1, {
  type: instant messaging,
  protocol: SIP/SIMPLE,
  duration: 5 minutes
});
```

Conferencing SaaS A (running on PaaS)
Conference Orchestration
Broker
Conferencing IaaS Handler
Dial-in Signaling (IaaS A)
Audio Mixing (IaaS B)
Instant Messaging (IaaS C)
1
2 Determine necessary substrates and composition template
3 Discover related SubaaSs
4 Signaling -> IaaS A, Audio Mixer -> IaaS B,
5 Create & activate signaling substrate
6 Create & activate substrate
7 Return substrate ID (S1)
8 Signaling substrate created (S1)
9 Create & activate Audio Mixer substrate
10 Create & activate substrate
11 Return substrate ID (A1)
12 Audio Mixer substrate created (A1)
13 Map new conference ID from substrates' IDs: C1 = (S1, A1)
14 Conference created (C1)
After 30 minutes ...
15
16 Discover related SubaaS
17 Instant Messaging -> IaaS C
18 Create & activate Instant Messaging substrate
19 Create & activate substrate
20 Return substrate ID (T1)
21 Instant Messaging ubstrate created (T1)
22 Modify orchestrated conference
Associate T1 with running conference: C1 = (S1, A1, T1)
23 Add conference C1's participants in substrate T1
24 Add conference C1's participants in substrate T1
25 Return Ack (Participants added)
26 Return Ack (Participants added)
27 Instant Messaging added to conference C1

FIGURE 3. Conference creation and modification steps.

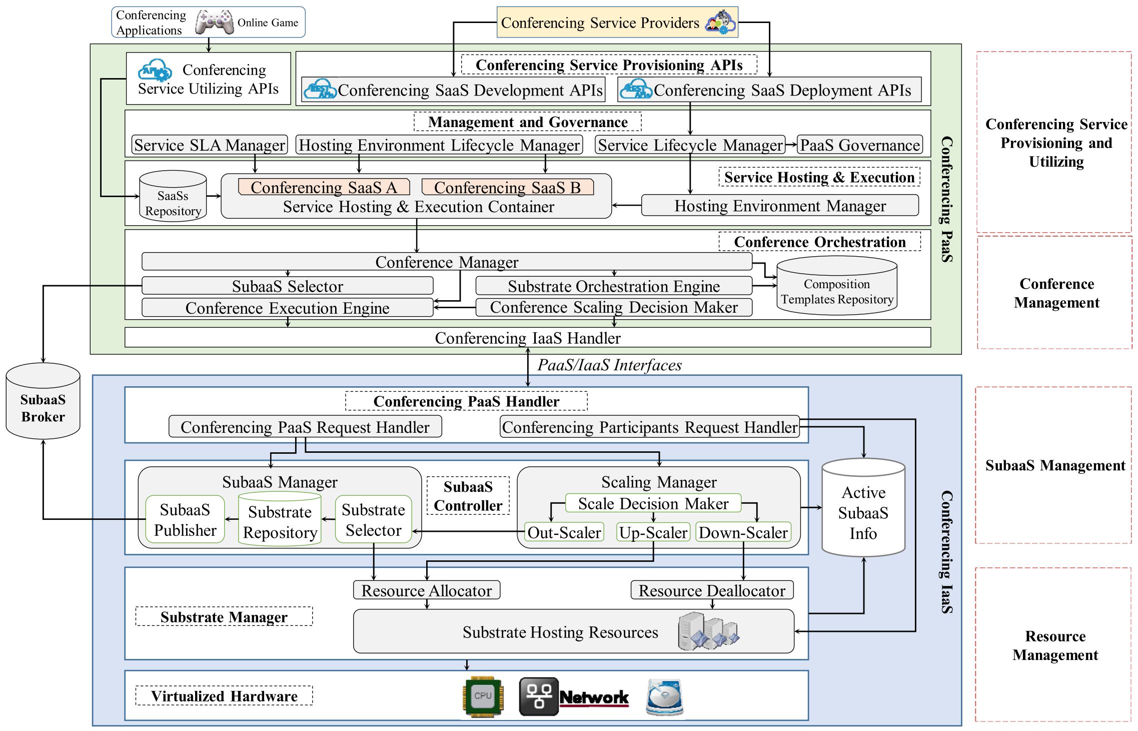

Conferencing Applications
Online Game
Conferencing Service Providers
Conferencing Service Utilizing APIs
Conferencing Service Provisioning APIs
Conferencing SaaS Development APIs
Conferencing SaaS Deployment APIs
Management and Governance
Service SLA Manager
Hosting Environment Lifecycle Manager
Service Lifecycle Manager
PaaS Governance
Service Hosting & Execution
SaaSs Repository
Conferencing SaaS A
Conferencing SaaS B
Service Hosting & Execution Container
Hosting Environment Manager
Conference Orchestration
Conference Manager
SubaaS Selector
Substrate Orchestration Engine
Composition Templates Repository
Conference Execution Engine
Conference Scaling Decision Maker
Conferencing IaaS Handler
Conferencing PaaS
PaaS/IaaS Interfaces
SubaaS Broker
Conferencing PaaS Handler
Conferencing PaaS Request Handler
Conferencing Participants Request Handler
SubaaS Manager
SubaaS Publisher
Substrate Repository
Substrate Selector
SubaaS Controller
Scaling Manager
Scale Decision Maker
Out-Scaler
Up-Scaler
Down-Scaler
Active SubaaS Info
Resource Allocator
Resource Deallocator
Substrate Manager
Substrate Hosting Resources
Virtualized Hardware
CPU
Network
Conferencing IaaS
Conferencing Service Provisioning and Utilizing
Conference Management
SubaaS Management
Resource Management

FIGURE 4. Implementation architecture.

The *SubaaS Selector* chooses the most suitable conferencing SubaaS from the SubaaS Broker, given the substrate requirements. The discovery mechanism and the interfaces between these two are reused from the existing work [32]. The *Substrate Orchestration Engine* uses the chosen template to compose the selected substrates and deploy it in the *Conference Execution Engine* that hosts the running conferences. The *Conference Scaling Decision Maker* monitors the running conferences and requests scaling when needed.

2) CONFERENCING IaaS

The *Conferencing PaaS Handler* includes two components: *Conferencing PaaS Requests Handler* and *Conferencing Participants Request Handler*. These are used to process the requests initiated by the PaaS and by the Conference Participants (i.e., the users of conferencing applications), respectively. These requests are of three types: (1) to create and activate a conference; (2) to scale a specific conference (e.g., change the conference size); and (3) to join and leave a conference. The first two are initiated by the PaaS while the third is used by the participants. Both handlers are implemented using REST APIs.

The conference creation and activation requests are sent to the *SubaaS Manager* in the SubaaS Controller component. The *SubaaS Manager* uses the *Substrate Selector* to choose the appropriate substrates for creating the new conference. Also, it uses the *SubaaS Publisher* to publish the existing SubaaSs to the Broker.

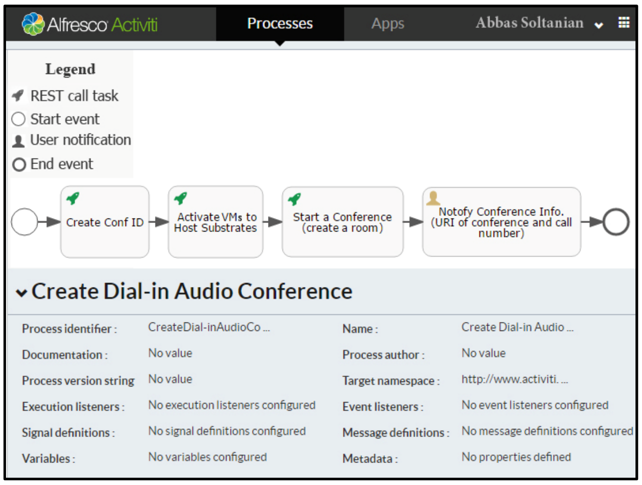
The conference scaling requests are forwarded to the *Scaling Manager* in the SubaaS Controller component. It relies on a *Scale Decision Maker* to decide how to scale the conference. The decision maker first fetches the information about the conference-related SubaaSs from the *Active SubaaS Info* (e.g., the IP of the hosting VM(s)/container(s) and the information of the server(s) hosting those substrates, such as available RAM, CPU, etc). Then, based on this information and the new scaling requirements, the decision maker determines which substrate(s) should be changed and how (i.e., scale up/out/down). It then instructs the appropriate component to do it (i.e., *Up-Scaler*,*Out-Scaler,* and *Down-Scaler*). For instance, if the requirement is to update an audio conference with 50 users to support 100 users and the current server hosting the audio mixing substrate does not have enough resources, the decision is to scale out the audio mixing substrate on another server. Thus, a new VM or container will create on another server to host the audio mixer substrate.

For the *Substrate Manager,* we use the OpenStack Compute (Nova) layer. It creates and updates VMs/Containers to host the running substrates. It allocates or deallocates resources based on the incoming requests from the *SubaaS Manager* and the *Scaling Manager*. It also keeps the *Active SubaaS Info* up-to-date after each operation.

B. PROTOTYPE

The prototype scenario includes a service provider offering dial-in audio conferencing service and a game application utilizing that service. It also includes the conferencing PaaS and two conferencing IaaSs – both providing dial-in signaling and audio mixer substrates.

In this prototype, the Cloud Foundry PaaS is used to provide the implementation of typical PaaS components. We also extend it to implement our novel component (i.e., *Conference Orchestration*). For *Substrate Orchestration Engine* and *Conference Execution Engine*, we use Activiti [38], a light-weight


Alfresco Activiti
Processes
Apps
Abbas Soltanian
Legend
REST call task
Start event
User notification
End event
Create Conf ID
Activate VMs to Host Substrates
Start a Conference (create a room)
Notofy Conference Info. (URI of conference and call number)

Create Dial-in Audio Conference

Process identifier :	CreateDial-inAudioCo ...	Name :	Create Dial-in Audio ...
Documentation :	No value	Process author :	No value
Process version string	No value	Target namespace :	http://www.activiti. ...
Execution listeners :	No execution listeners configured	Event listeners :	No event listeners configured
Signal definitions :	No signal definitions configured	Message definitions :	No message definitions configured
Variables :	No variables configured	Metadata :	No properties defined

FIGURE 5. Dial-in audio conference creation and activation workflow.

workflow and Business Process Management (BPM) platform. *Conference Manager* and *Conferencing IaaS Handler* are implemented using Express.js framework [39]. Advanced REST Client [40] is also used to simulate SaaS APIs invocation by the game.

For the conferencing IaaS, OpenStack [41] is used. *Conferencing PaaS Handler* is implemented as a Java application with REST-based APIs to communicate with the PaaS. The open source framework Asterisk [42] is used for signaling, media handling, and floor control substrates. To publish a SubaaS information, we implement a subset of the model proposed in [32]. Our published SubaaS information is shown in table 3. For the Scaling Manager, we use the existing resource allocation mechanism proposed in [35].

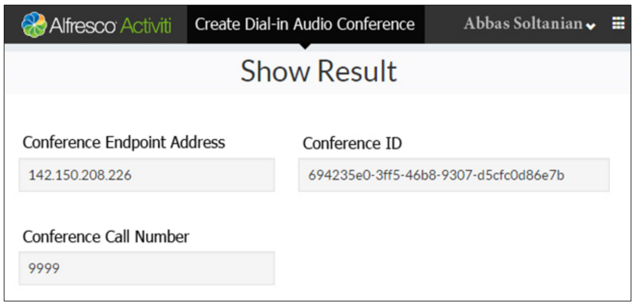
C. VALIDATION AND MEASUREMENTS

To validate our architecture, we run the implementation according to the steps in Fig. 3. Fig. 5 shows the Activiti orchestration process to create a dial-in audio conference. The workflow execution corresponds to steps 5 to 14 in Fig. 3. Implementing the dial-in audio conference service using Activiti proved the simplicity of service creation, which is very useful for non-expert developers. Indeed, while expert conferencing service providers can use offered APIs to create their provisioned services, non-expert providers can use an orchestration tool to provision their services. Fig. 6 shows the parameters sent back to the game application after the workflow execution.

Three experimental environments are considered for performance measurements: 1) A Non-Cloud Conferencing (NCC) environment, where resources are allocated beforehand. 2) A Monolithic IaaS Provider (MIP) environment, where an IaaS offers multiple substrates in a single SubaaS (i.e., the SubaaS is composed of multiple coupled substrates). Thus, the IaaS hosts all substrate instances on the same VM. This is the same if several SubaaSs from the same IaaS run on the same VM. 3) A Non-Monolithic IaaS Provider (NMIP)

TABLE 3. Published information of a SubaaS into the broker in our impementation.

Categories of Parameters	Example Values		
Type of Service	Signaling	Signaling Protocol	Acceptable signaling protocols (e.g. SIP)
		Conference Model	Acceptable conference models (e.g. Dial-in, Dial-out, Ad-hoc)
	Media Handling	Audio Mixer	Acceptable audio encodings (e.g. G.711)
		Video Mixer	Acceptable video encodings (e.g. H.264)
	Conference Control	Floor Control Policy	Supported control policies (e.g. round-robin)
		Floor Chairs	Maximum number of acceptable chairs (e.g. 5)
Service Endpoint	URI	The domain or IP of the service, along with the port number (e.g. http://audiomixer.com:263)	
Service Limit	Maximum number of accepting participants	Zero means there is no limit on maximum number of accepting participants	
Service Integration Templates	Workflows	The bpmn20.xml files that describes how this service can integrate with some other known services	



Show Result

Conference Endpoint Address: 142.150.208.226

Conference ID: 694235e0-3ff5-46b8-9307-d5cfc0d86e7b

Conference Call Number: 9999

FIGURE 6. Conference information which passed to the game application.

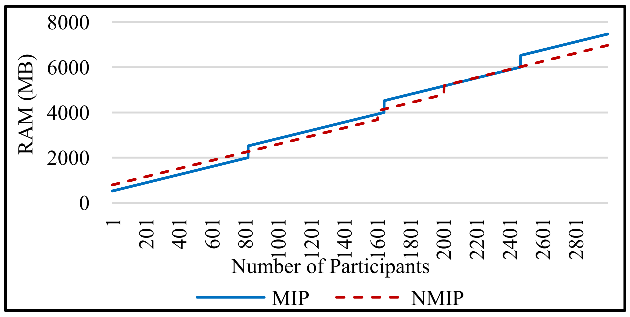


FIGURE 7. Resource allocation evaluation.

environment, where IaaS offers every single substrate as a separate service. In NMIP, the IaaS hosts substrate instances on separate VMs.

The following four metrics are used: (1) Resource Allocation – the total amount of allocated resources, such as memory and CPU, to accommodate all participants, (2) Scale Time – the time to add resources to scale the conference, (3) Conference Start Time – the time to get a conference ready upon the receipt of a request and (4) Participant Joining Time – the time to add a participant to a running conference.

To analyze the allocated resources, we consider a conferencing application with considerable fluctuation. A good example of such application is a massively multiplayer online game (MMOG) which offers the audio/video conferencing. This kind of application may include thousands or even millions of players who share their audio and video in the logic of the game. For example, the study in [43] reported that the number of users in World of Warcraft (a famous online game) fluctuates between 1.5 and 2.5 million over 10 hours. Fig. 7 shows the allocated amount of memory (i.e., RAM) for a conference, when the number of participants fluctuates between 1 and 3000. To simulate this fluctuation, we increase the conference size by 200 participants every 10 minutes. The results are based on the observed resource usage per participant. The Scale Decision Maker in IaaSs scales the VMs up and out while maintaining the QoS requirements. Two QoS requirements are considered: 1) the end-to-end delay which includes the audio and video mixing time should not take more than 400 msec and 2) the amount of allocated resources should be minimized. The resource allocation algorithm in [35] is designed for this purpose and reused for the prototype. The reference also discusses audio and video mixer placement and the placement effects on the QoS. This discussion is therefore not repeated here.

In NCC, there are always some idle and non-utilized resources because of upfront resource provisioning. Hence, we do not show the NCC allocated resources in Fig. 7. As it is depicted in this figure, MIP scales better than NMIP (i.e., it allocates fewer resources) for smaller conferences whereas NMIP wins for bigger conferences. In NMIP, the substrates are hosted on separate VMs. Thus, for smaller conferences, it leads to more VMs and more non-utilizable resources (e.g., the resources consumed by the operating system) than in MIP. The bigger the size of a conference, the more resources the substrates required to perform well. However, they do not require the same thing; e.g., a signaling substrate may need less extra resources than the mixer because it is only used in the first phase of the conference. In MIP, because of having monolithic SubaaS, the rate of adding resources is the same for all substrates. This results in more scaling out decisions and therefore more VMs. Indeed, by applying the allocation algorithm in [35], the resources exceed its maximum extra amount for scaling up, which makes scaling out a better decision. By contrast, in NMIP, the resources are allocated to each substrate based on their need, resulting in less scaling out decisions. This makes NMIP achieve better scalability because of the fewer number of VMs and better resource utilization than in MIP.

Regarding CPU usage, we used a 2.6 GHz single core CPU for each VM. The CPU utilization per VM fluctuated between 20% and 80% for each VM in both scenarios. This fluctuation is based on the number of users that are connected to the VM. This shows that VMs' resources are not fully used, in both MIP and NMIP. Therefore, CPU utilization for small conferences is better in MIP, since it has a fewer number of VMs to accommodate users in comparison with NMIP. Similarly, when the size of the conference is big, NMIP has better results because of its fewer VMs usage.

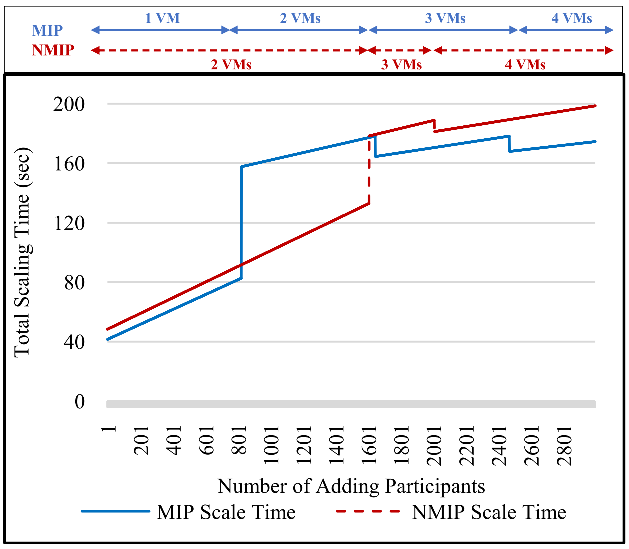


FIGURE 8. **Total time for scaling the size of a conference with single participant to a conference with 2 up to 3000 participants.**

For the Scale Time metric, we observe the scaling performance of the system under two conditions. The first condition demonstrates the behavior of the system when the conference starts with the minimum required amount of resources, i.e., the least possible substrate instances (Fig. 8). The second condition demonstrates the behavior of the system under resource over-provisioning situation, i.e., the conference starts with more substrate instances than required (Fig. 9). Note that the second experiment is exclusively aimed for demonstrating the impact of the number of substrates on the scaling time. Therefore, in that experiment, the SLA violations are not taken into account; i.e., the amount of allocated resources is not minimized. The provided set of experiments helps the conference service providers to evaluate the tradeoff between over-provisioning and (sub)optimal substrate allocation.

The scaling time under the first condition for both MIP and NMIP scenarios are depicted in Fig. 8. In this experiment, we first run a conference with one participant. This conference starts with the minimum required resources (i.e., one VM in MIP and two VMs in NMIP). By increasing the size of

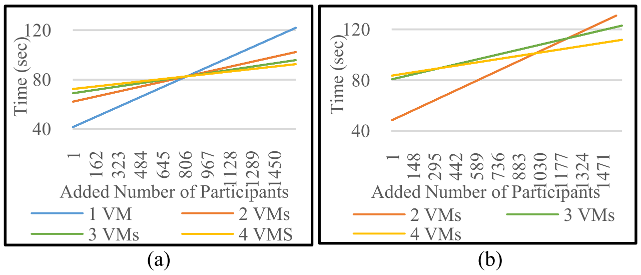


FIGURE 9. **Conference scaling time by having different number of VMs for (a) MIP and (b) NMIP.**

the conference, the required resources are added to the existing VMs (i.e., those hosting the substrates). If the required QoSs cannot be satisfied by adding resources to the existing VMs, (e.g., the end-to-end delay is more than 400 msec), the Scaling Manager in the IaaS starts new VMs for hosting another instance of required substrates [35]. We scale the size of the conference between 1 and 3000 participants in this experiment. The scaling time accounts for several parameters, including the time for creating a new VM, the time for adding resources (i.e., RAM in this experiment) to the existing VMs and reconfiguring the system (e.g., updating the list of available audio mixers), and the time for adding all the participants. Basically, the scaling time from 10 to 100 participants, for instance, is the total time for moving from a running conference with 10 participants to a running conference with 100 participants.

As shown in Fig. 8, for adding a large number of participants, the scaling time in MIP is lower than that of NMIP. The main reason is, as discussed earlier, MIP creates more instances/VMs than NMIP when the number of participants is large. This makes MIP able to add participants in parallel to several substrate instances/VMs. Besides, although reconfiguring the system with more instances has some overhead, the gain from load balancing makes the scaling time in MIP lower than in NMIP.

Also, MIP gives better results when a limited number of participants is added (scaling between 1 through ~825 in this graph). Adding resources to the existing VMs in both scenarios takes the same time, i.e., it is the exact same process. However, the overhead time of reconfiguring the system with more VMs in NMIP leads to having a longer scaling time. Once MIP starts to create new VMs, the time for creating these VM leads to an increase in the scaling time in MIP (in the middle part of the figure). Based on the results, no matter how many VMs are created, it does not noticeably affect the scaling time in MIP and NMIP because several VMs can be created in parallel.

The case of overprovisioning (Fig. 9) shows that, in both MIP and NMIP, the scaling time for a limited number of participants (up to ~800 participants) is less when the number of VMs is less. In contrast, when the number of participants to be added is large, having a conference that is hosted on more VMs results in less scaling time because the participants can join multiple instances/VMs in parallel. Therefore, we can conclude that having more resources does not always lead to having less scaling time in the conferencing domain. In fact, in conferencing, the collaboration between different substrate instances hosted on different VMs causes some overhead. Although increasing the number of substrate instances and balancing the loads between them leads to some saving in scaling time, the overhead of reconfiguring the system might be more than the gain.

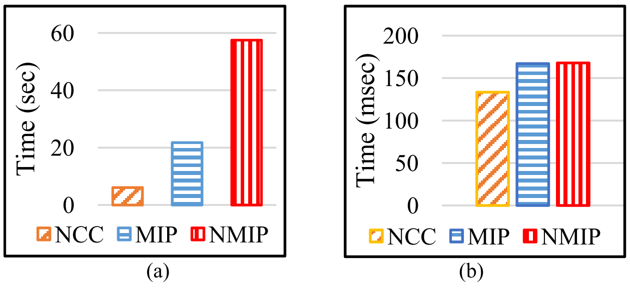


FIGURE 10. Average (a) conference start time (b) participant joining time.

Fig. 10(a) compares the conference start time in the three studied environments (i.e., NCC, MIP, and NMIP). It shows that NCC takes the least time to start a new conference, which is obvious due to the absence of virtualization overhead. And, since in NMIP the substrate instances are hosted on separate VMs and they need to connect to each other over the network, it takes more time than it does in MIP. However, since starting a conference happens just once, this time is endurable in the Cloud scenarios. Participant joining time is also the least in the NCC as shown in Fig. 10(b). Cloud-based scenarios take more time because of the notification overhead between IaaSs, PaaS and the game server. However, this time length remains acceptable (can be seen as the waiting time to join the conference) and is not noticeable by end users. In addition, the participant joining time of the two cloud-based scenarios are close as IaaSs can notify PaaS in parallel.

In conclusion, although in cloud-based scenarios, the conference start time and the participant joining time are more than those in NCC, the cloud-based conferencing architecture helps to scale the system easily and avoid the over-provisioning or under-provisioning of resources. The results of scaling the time and allocated resources help the conferencing service providers with better provisioning of their services. For instance, MIP is a better choice for provisioning small conferencing services (e.g., to support 300 users) as it results in less resource usage and less scaling time than in NMIP. However, for a conferencing service with 1200 users, NMIP gives better scaling time and lower resource consumption. In the case of big scenarios (e.g., 3000 users or more), there is a tradeoff between using less resources (i.e., NMIP) and having less scaling time (i.e., MIP).

VI. CONCLUSION

A novel holistic architecture for multimedia conferencing is proposed in this paper. This architecture simplifies the provisioning of the conferencing services for expert and non-expert service providers. It also supports scaling the running conferences in an elastic manner. The implemented prototype and the run experiments show the feasibility and validation of the proposed architecture. The measurement analysis provides important findings, together with valuable recommendations to the conferencing service providers, to suggest them the most appropriate alternative for the deployment of their service.

REFERENCES

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: Towards a cloud definition," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, pp. 50–55, 2008.

[2] R. H. Glitho, "Cloud-based multimedia conferencing: Business model, research agenda, state-of-the-art," in *Proc. IEEE 13th Conf. Commerce Enterprise Comput. (CEC)*, Sep. 2011, pp. 226–230.

[3] M. Jacobs and P. Leydekkers, "Specification of synchronization in multimedia conferencing services using the TINA lifecycle model," *Distrib. Syst. Eng.*, vol. 3, no. 3, p. 185, 1996.

[4] A. F. B. Alam, A. Soltanian, S. Yangui, M. A. Salahuddin, R. Glitho, and H. Elbiaze, "A cloud platform-as-a-service for multimedia conferencing service provisioning," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2016, pp. 289–294.

[5] L. Coyne, T. Hajas, M. Hallback, M. Lindström, and C. Vollmar, *IBM Private, Public, and Hybrid Cloud Storage Solutions*. Armonk, NY, USA: IBM Redbooks, 2016.

[6] B. Sotomayor, R. S. Montero, I. M. Llorente, and I. Foster, "Virtual infrastructure management in private and hybrid clouds," *IEEE Internet Comput.*, vol. 13, no. 5, pp. 14–22, Oct. 2009.

[7] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Univ. California, Irvine, Irvine, CA, USA, 2000, vol. 7.

[8] Q. Z. Sheng, X. Qiao, A. V. Vasilakos, C. Szabo, S. Bourne, and X. Xu, "Web services composition: A decade's overview," *Inf. Sci.*, vol. 280, pp. 218–238, Oct. 2014.

[9] "Web service glossary," World Wide Web Consortium (W3C), Cambridge, MA, USA, Tech. Rep. W3C, 2004.

[10] T. Yu and K.-J. Lin, "Service selection algorithms for composing complex services with multiple QoS constraints," in *Service-Oriented Computing*, B. Benatallah, F. Casati, and P. Traverso, Eds. Berlin, Germany: Springer, 2005, pp. 130–143.

[11] W. Zeng, Y. Zhao, and J. Zeng, "Cloud service and service selection algorithm research," in *Proc. 1st ACM/SIGEVO Summit Genet. Evol. Comput.*, 2009, pp. 1045–1048.

[12] J. Li, R. Guo, and X. Zhang, "Study on service-oriented cloud conferencing," in *Proc. 3rd IEEE Int. Conf. Comput. Sci. Inf. Technol. (ICCSIT)*, Jul. 2010, pp. 21–25.

[13] R. L. Grossman, "The case for cloud computing," *IT Prof.*, vol. 11, no. 2, pp. 23–27, Mar. 2009.

[14] P. Rodríguez, D. Gallego, J. Cerviño, F. Escribano, J. Quemada, and J. Salvachúa, "VaaS: Videoconference as a service," in *Proc. 5th Int. Conf. Collaborative Comput., Netw., Appl. Worksharing*, 2009, pp. 1–11.

[15] F. Taheri, J. George, F. Belqasmi, N. Kara, and R. Glitho, "A cloud infrastructure for scalable and elastic multimedia conferencing applications," in *Proc. 10th Int. Conf. Netw. Service Manage. (CNSM)*, 2014, pp. 292–295.

[16] Y. Feng, B. Li, and B. Li, "Airlift: Video conferencing as a cloud service using inter-datacenter networks," in *Proc. 20th IEEE Int. Conf. Netw. Protocols (ICNP)*, Nov. 2012, pp. 1–11.

[17] R. Cheng, W. Wu, Y. Lou, and Y. Chen, "A cloud-based transcoding framework for real-time mobile video conferencing system," in *Proc. 2nd IEEE Int. Conf. Mobile Cloud Comput., Services, Eng. (MobileCloud)*, Apr. 2014, pp. 236–245.

[18] J. Liao, C. Yuan, W. Zhu, and P. A. Chou, "Virtual mixer: Real-time audio mixing across clients and the cloud for multiparty conferencing," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process. (ICASSP)*, Mar. 2012, pp. 2321–2324.
[19] C. Vecchiola, X. Chu, and R. Buyya, "Aneka: A software platform for.NET-based cloud computing," *High Speed Large Scale Sci. Comput.*, vol. 18, no. 3, pp. 267–295, 2009.
[20] *Cloud Foundry Overview*. Accessed: Nov. 4, 2015. [Online]. Available: http://docs.cloudfoundry.org/concepts/overview.html
[21] X. Wen, G. Gu, Q. Li, Y. Gao, and X. Zhang, "Comparison of open-source cloud management platforms: OpenStack and OpenNebula," in *Proc. 9th Int. Conf. Fuzzy Syst. Knowl. Discovery (FSKD)*, 2012, pp. 2457–2461.
[22] O. Litvinski and A. Gherbi, "Openstack scheduler evaluation using design of experiment approach," in *Proc. 16th IEEE Int. Symp. Object/Compon./Service-Oriented Real-Time Distrib. Comput. (ISORC)*, Jun. 2013, pp. 1–7.
[23] S. U. R. Malik, S. U. Khan, and S. K. Srinivasan, "Modeling and analysis of state-of-the-art VM-based cloud management platforms," *IEEE Trans. Cloud Comput.*, vol. 1, no. 1, p. 1, Aug. 2013.
[24] N. Milanovic and M. Malek, "Current solutions for Web service composition," *IEEE Internet Comput.*, vol. 8, no. 6, pp. 51–59, Dec. 2004.
[25] A. Urbieta, G. Barrutieta, J. Parra, and A. Uribarren, "A survey of dynamic service composition approaches for ambient systems," in *Proc. Ambi-Syst. Workshop Softw. Organisation Monitor. Ambient Syst.*, 2008, p. 1.
[26] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, Oct. 2003.
[27] W. Chareonsuk and W. Vatanawood, "Formal verification of cloud orchestration design with TOSCA and BPEL," in *Proc. ECTI-CON*, Jun. 2016, pp. 1–5.
[28] D. Martin *et al.*, *OWL-S: Semantic Markup for Web Services*, document 04–2007, 2004, vol. 22.
[29] A. Kim, M. Kang, C. Meadows, E. Ioup, and J. Sample, *A Framework for Automatic Web Service Composition*, DTIC document ADA499917, 2009.
[30] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski, "Introducing STRATOS: A cloud broker service," in *Proc. IEEE CLOUD*, vol. 12. Jun. 2012, pp. 891–898.
[31] S. Yangui, I.-J. Marshall, J.-P. Laisne, and S. Tata, "CompatibleOne: The open source cloud broker," *J. Grid Comput.*, vol. 12, no. 1, pp. 93–109, 2014.
[32] J. George, F. Belqasmi, R. H. Glitho, and N. Kara, "A substrate description framework and semantic repository for publication and discovery in cloud based conferencing," in *Proc. CSWS*, 2013, pp. 41–44.
[33] M. Garriga, C. Mateos, A. Flores, A. Cechich, and A. Zunino, "RESTful service composition at a glance: A survey," *J. Netw. Comput. Appl.*, vol. 60, pp. 32–53, Jan. 2016.
[34] S. Fu, J. Liu, X. Chu, and Y. Hu, "Toward a standard interface for cloud providers: The container as the narrow waist," *IEEE Internet Comput.*, vol. 20, no. 2, pp. 66–71, Apr. 2016.
[35] A. Soltanian, M. A. Salahuddin, H. Elbiaze, and R. Glitho, "A resource allocation mechanism for video mixing as a cloud computing service in multimedia conferencing applications," in *Proc. 11th Int. Conf. Netw. Service Manage. (CNSM)*, 2015, pp. 43–49.
[36] A. B. Johnston and D. C. Burnett, *WebRTC: APIs and RTCWEB Protocols of the HTML5 Real-Time Web*. St. Louis, MO, USA: Digital Codex LLC, 2012.
[37] *Activiti*. Accessed: Nov. 7, 2016. [Online]. Available: http://activiti.org/
[38] M. Geiger, S. Harrer, J. Lenhard, M. Casar, A. Vorndran, and G. Wirtz, "BPMN conformance in open source engines," in *Proc. IEEE Symp. Service-Oriented Syst. Eng. (SOSE)*, Mar. 2015, pp. 21–30.
[39] *Express—Node.js Web Application Framework*. Accessed: Mar. 25, 2017. [Online]. Available: https://expressjs.com/
[40] *Advanced REST Client*. Accessed: Feb. 27, 2016. [Online]. Available: https://chrome.google.com/webstore/detail/advanced-rest-client/hgmloofddffdnphfgcellkdfbfbjeloo
[41] *OpenStack Open Source Cloud Computing Software*. Accessed: Feb. 26, 2016. [Online]. Available: https://www.openstack.org/
[42] *Asterisk*. Accessed: Sep. 11, 2016. [Online]. Available: http://www.asterisk.org/
[43] V. Nae, R. Prodan, and T. Fahringer, "Cost-efficient hosting and load balancing of massively multiplayer online games," in *Proc. 11th IEEE/ACM Int. Conf. Grid Comput. (GRID)*, Oct. 2010, pp. 9–16.

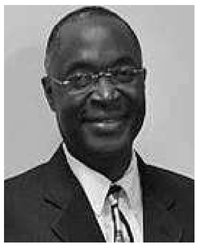
ABBAS SOLTANIAN received the M.Sc. degree in information technology from the Sharif University of Technology, Iran. He is currently pursuing the Ph.D. degree in information and systems engineering with Concordia University, Montreal, QC, Canada. His current research interests include cloud computing, multimedia conferencing, and cloud resource optimization.

FATNA BELQASMI received the M.Sc. and Ph.D. degrees in electrical and computer engineering from Concordia University, Montreal, QC, Canada. She was a Research Associate with Concordia University and a Researcher with Ericsson Canada. She was a part of the IST Ambient Network Project (a research project sponsored by the European Commission within the Sixth Framework Programme—FP6). She was a Research and Development Engineer with Maroc Telecom, Morocco. She is currently an Associate Professor with Zayed University, Abu Dhabi, United Arab Emirates. Her research interests include next-generation networks, service engineering, distributed systems, and networking technologies for emerging economies.

SAMI YANGUI received the M.Sc. degree in computer science from the University of Tunis El-Manar, Tunisia, in 2010, and the Ph.D. degree in computer science from the Telecom SudParis, Institut Mines-Telecom, France, in 2014.

He is an Associate Professor with the Institut National des Sciences Appliquées, Toulouse, France. He is a member of the CNRS LAAS research team. He is involved in different aspects related to these topics, such as cloud/fog computing, network functions virtualization, and content delivery networks. He is also involved in different European and international projects, and standardization efforts. He published several scientific papers in high-ranked conferences and journals in his field of research. His research interests include distributed systems and architectures, service-oriented computing and Internet of Things. He also served on many program and organization committees of international conferences and workshops.

MOHAMMAD A. SALAHUDDIN (S'09–M'15) received the Ph.D. degree in computer science from Western Michigan University, Kalamazoo, MI, USA, in 2014. He was a Post-Doctoral Fellow with the Université du Québec à Montréal and a Visiting Scientist with Concordia University, Montréal, QC, Canada. He is currently a Post-Doctoral Fellow with the David R. Cheriton School of Computer Science, University of Waterloo, Waterloo, ON, Canada. His research interests include wireless sensor networks, QoS and QoE in vehicular ad hoc networks (WAVE, IEEE 802.11p, and IEEE 1609.4), Internet of Things, content delivery networks, software-defined networking, network functions virtualization, and cloud computing. He serves as a Technical Program Committee Member of international conferences and a reviewer of various peer-reviewed journals, magazines, and conferences.



ROCH GLITHO received the M.Sc. degrees in business economics from the University of Grenoble, France, and in pure mathematics and computer science from the University of Geneva, Switzerland, and the Ph.D. (Tekn. Dr.) degree in tele-informatics from the Royal Institute of Technology, Stockholm, Sweden. He was with industry and has held several senior technical positions (e.g., senior specialist, principal engineer, and expert) at Ericsson in Sweden and Canada. His industrial experience includes research, international standards setting, product management, project management, systems engineering, and software/firmware design. He is currently an Associate Professor and the Canada Research Chair with Concordia University. He is also an Adjunct Professor at several other universities, including Telecom Sud Paris, France, and the University of Western Cape, South Africa. He served as an IEEE Distinguished Lecturer, the Editor-In-Chief of the *IEEE Communications Magazine*, and the Editor-In-Chief of the IEEE COMMUNICATIONS SURVEYS AND TUTORIALS JOURNAL.

HALIMA ELBIAZE received the B.S. degree in applied mathematics from the University of MV, Morocco, in 1996, the M.Sc. degree in telecommunication systems from the Université de Versailles in 1998, and the Ph.D. degree in computer science from the Institut National des Télécommunications, Paris, France, in 2002. In 2003, she joined the Department of Computer Science, Université du Québec à Montréal, Montreal, QC, Canada, where she is currently an Associate Professor. She is the author or co-author of many journals and conference papers. Her research interests include network performance evaluation, traffic engineering, and quality of service management in optical and wireless networks.