VaaS: Videoconference as a Service

Pedro Rodríguez, Daniel Gallego, Javier Cerviño, Fernando Escribano, Juan Quemada, Joaquín Salvachúa
Departamento de Ingeniería de Sistemas Telemáticos, Universidad Politécnica de Madrid,
Avda. Complutense 30, Ciudad Universitaria,
28040 Madrid, Spain
{prodriguez, dgallego, jcervino, fec, jquemada, jsalvachua}@dit.upm.es

Abstract — **Internet is a place nowadays where interoperating services are offered which can be integrated or mashed up in order to fulfill user demands. This paper proposes a way to offer videoconference as a web service over an interface which can be used by third parties to enrich their applications. This interface includes a security mechanism supporting delegated authorization to allow integration into third party's environments. Via this interface virtual rooms are provided where users can collaborate with audio, video, shared applications, IM, etc. An implementation of these concepts is described, including performance figures and validation results. We would finally like to stress that this architecture has been defined to support a scalable cloud computing service over the Internet.**

Cloud computing; Videoconferencing; Web; Real-Time Collaboration; SOA; ROA.

I. INTRODUCTION

A wide variety of services is accessed nowadays via open interfaces, such that their functionalities can be combined offering an extra value to final users. Today we can share our photos in Flickr, with Twitter using widgets merged into iGoogle. Why not integrating collaborative rooms where we can work together with others. Internet services are becoming more and more usual, and mash-ups and combinations which generate added value are becoming a must in the Internet.

In this paper, we propose a new interface for integrating collaborative videoconferencing as another service in those mash-ups. It would provide to any web application a videoconferencing service, executable from the browser. This way, users can communicate among themselves easily.

Our proposal meets the following requirements:

- User management is taken care of by third parties' applications while it leaves the authorization to them.
- It focuses on conference rooms, where users meet to collaborate.
- The security requirements have been designed to allow integration into third party's applications.
- The interface provides some Quality of Service (QoS) inside each room to every application (or consumer services) used.

In other words, this interface aims to transform a traditional telecommunication service, such as videoconference, into a resource that will be used by third parties. Furthermore, this approach enables the transformation of a standard client-server service into a Cloud Computing service which provides to users access to a collaborative environment including audio, video and shared applications. This architecture has been named "Nuve".

The structure of the paper is as follows: the next section gives a summary of work closely related to this topic. Section 3 aims to place this work in context while the following two sections introduce the conceptual model and define the operations of this interface. Section 6 describes the designed security mechanism to authenticate requests to the interface. Finally, the last two sections present the results obtained and detail the conclusions drawn from the work.

II. RELATED WORK

A variety of videoconferencing applications exist in Internet such as [1], [2], [3], [4], etc. They all enable collaborative videoconferencing with more or less functionality. However, none of them provides a way for third parties' applications to take advantage of those resources. We feel that videoconference and real-time collaboration tools in general can be a very powerful complement for many existent applications.

Our idea is based on telecommunication operator's Data Centers. While typical software and Web companies, such as Google and Microsoft, are focusing their attention on terms like Cloud Computing when in fact, ISPs have Data Centers more sophisticated than they have. They can use these to house third parties' services, or, in the other hand, to restructure their own services.

In our proposal, Nuve, the evolution began from the Marte 3.0 [5] client-server web-conferencing application developed by our work group. It is based on rooms, and it provides users with tools for performing web collaboration such as: video, audio, instant messaging and desktop sharing.

III. VIDEOCONFERENCE AND CLOUD COMPUTING

There exist many videoconferencing applications but users do not use them very frequently and do not extract all the potential they have. This is probably because they cannot be easily integrated in existing user environments. This is why we think that by offering collaborative videoconferencing as a Cloud Computing service, users will integrate it more easily into their environments.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8285
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8285

In [7], it is said that Cloud Computing distinguishes itself from other computing paradigms in the following aspects:

- *User-centric interfaces:* using Web browsers. Nuve allows users to connect videoconference rooms through a Flex application executing in the web browser. This application offers a common user interface for all of them.
- *On-demand service provisioning:* this paradigm provides resources and services for users on demand. Nuve provides rooms to the services so users can use it.
- *QoS guaranteed offer:* computing clouds guarantees bandwidth, latency and so on. Nuve guarantees a correct bandwidth to the services because it analyzes the connection state and it modifies the communication parameters to adapt it.
- *Autonomous System:* the cloud is autonomous and transparent to users. Our objective is for Nuve to become a videoconference room provider to higher level services.
- *Scalability and flexibility:* the architecture is flexible, so it can adapt and scale itself depending on the number of users. In the present version of Nuve, this property is not yet implemented, but in the future, Nuve pretends to be scalable and flexible using clusters.

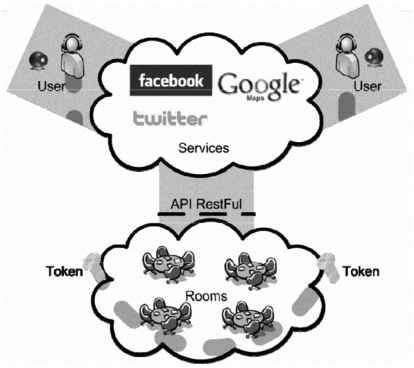

Fig. 1 Design of Videoconference as a Service

In addition, Nuve has some technological properties that also play an important role in Cloud Computing. These properties are described in [7]:

- *Virtualization:* in the present version there is only one virtual machine. However, in the future many virtual instances of Nuve could be started on demand, using similar services like Amazon EC2 [8].
- *Orchestration:* using APIs, an application can be the result of orchestrating a set of services. One of these services could be Nuve.
- *Web Service and Service Oriented Architecture:* Nuve offers an interface to control their resources, in other words, the videoconference rooms.
- *Web 2.0:* the Nuve user interface follows the philosophy of the Rich Internet Applications (RIA), using Flex applications to increase the usability.

In words of the NIST [6] "*Cloud computing is a pay-per-use model for enabling available, convenient, on-demand network access to a shared pool of configurable computing resources*". Thus, using REST interfaces to offer our Cloud Computing services seems to be natural.

Regarding the API architecture in this context, the objectives of our work were:

- Following the REST principles designing a generic, simple, flexible, reusable and stateless interface, useful to a multiconference service.
- Building a Cloud Computing oriented service that follows the principles we are going to explain later.
- Implementing a proof of concept.

The API developed will allow us to do all this things in the future.

IV. Conceptual Model

The main objective of Nuve architecture is to offer a videoconference service for third parties. As such, others will manage and control the communication while the system is the responsible of maintaining the communication alive and guarantee a Quality of Service. To understand this correctly, first we need to define a model based on resources and actors that will use those resources. In this section, we are going to describe the resources and their functionality inside the service.

TABLE 1
RESOURCES FROM THE CONCEPTUAL MODEL

Resource	Definition
Users	It represents the third parties' users who will communicate with others in each room.
Tokens	It is the ticket used to delegate the authorization of users to third parties' applications.
Services	They are the third party's applications that manage the rooms and give access to users in order to communicate in each of these rooms.
Rooms	They are the virtual spaces where the users will collaborate among themselves with video, audio and desktop sharing.

The conceptual model this architecture follows is that of a videoconference room provider (Fig. 1). We can define these rooms as meeting points where users will be able to establish conversations using audio, video and IM. Besides, they will be able to share applications executed in their own desktops. The management and control of these rooms is taken care of by other services. For this reason, they will be responsible for making these rooms accessible and giving users and other

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8285
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8285

services the needed authorizations (via tokens). In accordance with this, we can define four basic resources (TABLE 1). Each of those types of resources will be managed over the REST interface of Nuve with the standard HTTP operations: GET, POST, PUT and DELETE.

A. Users

A Nuve user is a person who has accessed a room and is communicating with other users from Nuve in the same room.

Every user can share his audio from his microphone or send the video obtained from his webcam. Besides, he can show or give control over his applications that are being executed in his computer. Also, he can chat to other users using the IM client incorporated in the room. All of these things are done using an application that is executing in the web browser through the user's Flash Player.

Using Flash Player as base for a videoconference application allows us to assert (based on [9]) that, in the great majority of desktop computers it is not necessary to install any kind of software, apart from the software that is already installed. To understand better this statement, we could say, for instance that any computer which has already accessed a video from Youtube, can access a Nuve videoconference without any problem.

Additionally, if a user wants to share applications executed in his computer, he must install the Java Virtual Machine. However, as we can see in [9], the Java Virtual Machine is usually installed in the great majority of computers.

In the real world, every participant in a meeting plays a different role so, in Nuve, there are three roles for users:

1) Observer

The observer user is present in the room and can see and listen to everything that is happening, but he cannot take part.

2) Participant

This role represents those users that can speak with the rest sending their video, audio, IM and desktop applications.

3) Administrator

Administrator role allows changing other users control over their videos, audios, IM and applications. Also, administrators can change the mode with which the users show themselves to everyone in the room.

B. Rooms

Rooms are the main resource in which the Nuve service model is based on. As we defined previously, a room is a virtual space where the users can communicate among themselves in meetings. Depending on the character of the meeting, a Nuve room may be equivalent in the real world to a company meeting room, an auditorium, a round table, a university class or the living room of a house where friends gather to speak about daily events. The objective of the meeting is defined by the service and his users. The Nuve aim is to guarantee bandwidth and Quality of Service in all the rooms.

The possible communication channels among users are voice, video, instant messaging (IM) and desktop sharing.

C. Services

This resource represents those consumer services that want to use the rooms provided by Nuve. To understand it well, we can use Facebook as an example of a service as it could be arranged for its users to participate in a Nuve conference. When we add a new service in Nuve, in fact we create a shared key between them. This key is used by the service to make calls to REST methods from the API. From now on, the service can create, edit and delete rooms, giving access to its users or denying it.

D. Tokens

Although we will see them in detail in the API security section, we can define the tokens here as a resource necessary for the whole user's authentication between the service provider (Nuve) and the consumer. A token represents the entrance key for a user in an existing session in Nuve. The service is the one who requests the token through the REST interface. Finally, it provides the token to the user so he or she can connect to the session.

V. NUVE API

In this section we will explain the REST architecture done in this work (illustrated in Fig. 2), which, as we said before, is based on four resources: users, rooms, services and tokens. We will see which HTTP operations are enabled for each one and what is expected to send and received in each call.

Consumer services will send HTTP requests to this interface, asking for the creation/deletion of rooms, giving access to users and retrieving information about conferences. End users of these services could take the role of administrators who will create, delete and modify rooms, or it could be done by the service automatically. Even third parties could create a wrapper in order to offer different kinds of services that would include videoconferencing rooms, and would manage them through protocols like SOAP, XMPP, etc.

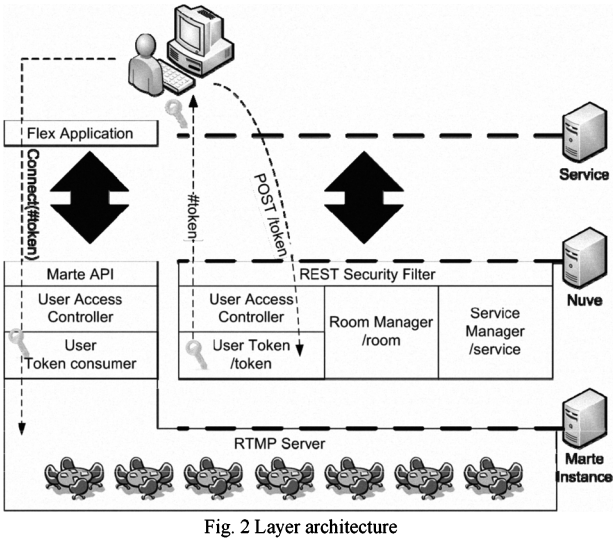

Fig. 2 Layer architecture

In order to support the usage of rooms by the services, the information about resources should be submitted in different ways. In our case, we have chosen the next three data structures

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8285
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8285

that are the most used nowadays in the World Wide Web: JSON [10] (which is the JavaScript serialization of objects, very important for objects implemented with this architecture), XML (that is a markup language with data structures compatible with JSON and is implemented in almost all application servers), and at last, HTML which is used mainly for making data representation easier to other services.

The API has been designed according to the recommendations from [11] for the purpose of making a good use of the REST philosophy. As it is usually recommended when designing this kind of APIs, our work proposes different commands using HTTP methods (GET, POST, PUT and DELETE) on the next four resources:

A. User

It represents users, providing different services for obtaining a list of them as well as information about one in particular.

We could get information about a user sending a GET HTTP request to a URL with the next structure: /room/{id}/user/{username}, being {id} the room identifier to which the user is supposed to be connected and {username} the name that the user has in this room. The response can be sent back in any of these formats: XML, JSON and HTTP. An example for this kind of communication would start with:

```
GET /room/321/user/Bob HTTP/1.1
Accept: application/xml
[Security info]
[CRLF]
```

The response would have the next structure:

```
HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: 60
[CRLF]
<user><username>Bob</username><role>participant</r
ole><status>online</status></user>
```

Furthermore, we could also ask for the entire list of connected users by sending a GET command to the URL /room/{id}/user/. We could even throw a user out from the room by sending a DELETE message to /room/{id}/user/{username}.

On this resource we cannot create users, although it could be useful if we would want to invite users to one room. However, since a user is going to connect directly to Nuve, we decided to let the implementation of this functionality to consumer services. A use case could be a scenario in which a user would send an email to other person with the link to the room. If the other user would click the link it would take him to the room.

In short, and referring to tables 1 and 2, it is possible to request information in XML, JSON or HTML format of the resource which represents the user who is connected to the Nuve's room.

B. Room

It represents the space in which users can collaborate and share their video, audio and data. We can perform different operations such as create, modify, delete, list rooms and even give access to users. As we said before, this is the main resource of Nuve and, as such, it is important for us to have perfect control and get detailed information in various formats.

This resource designed to be directly managed by consumer services or by users that could interact with it through their respective services.

Regarding the available operations, we will start with those of them that are read-only, which are the same that in the case of the resource "user".

Any service could request information about any of the rooms which belong to it. For instance, by sending a method GET to /room/ if it wants to get a list of all of its rooms, or to /room/{id} if it wants information about only one of them. The information returned by this way can be represented in XML, JSON or HTML format, and an example of the HTML one is below:

```
<noscript>
<object  id="MarteRoom">
<param name="movie" value="Marte.swf"/>
<param name="quality" value="high"/>
...
<embed src="Marte.swf"
     quality="high"
     type="application/x-shockwave-flash"
     ...
     play="true"/>
</object>
</noscript>
```

This example shows part of the code of a web page which would be useful to include in the Nuve client that, as we mentioned before, is a Flash application.

Regarding the "write" requests, the first one would be the creation of a conferencing room that, as it usually occurs in REST architectures, is accessible through a POST method to the collection URL. In this case an example for this would be:

```
POST /room HTTP/1.1
Content-Type: application/json
Content-Length: 26
[Security info]
[CRLF]
{"room":{"name":"MyRoom"}}
```

The information sent to the server could be either in XML format or in JSON (that is the case of this example):

The response of the server would be the next:

```
HTTP/1.1 301 Moved Permanently
Content-Type: application/json
Content-Length: 26
Location: /room/MyRoom
[CRLF]
{"room":{"name":"MyRoom"}}
```

As we see in this example, the response in REST models to the request for the creation of a resource by means of a POST method is a message of the type "Moved Permanently", and the

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8285
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8285

response includes among its headers one that shows the URL where the information of the room is located.

The other writing operations are used to update and delete rooms. The first one is a PUT request sent to the URL of the room (in the example it would be /room/MyRoom) with new information about the resource. The response for this request would be a "200 OK".

The last option would be to delete the room. This can be done through the DELETE operation, as we can see at example:

```
DELETE /room/MyRoom HTTP/1.1
 [Security info]
[CRLF]
```

The response in the case of Nuve successfully removing the room would be a "200 OK".

C. Service

Nuve's API allows adding and removing services that are authorized to use its Rooms. Furthermore, we can ask for the entire list of services through a "root" service with special permissions. As in previous cases, to create a service we will use a POST operation to the URL of the resource collection (/service), and to delete an existent service we will do the same with the DELETE operation to the URL of the resource which we want to remove (/service/{service_id}). It is important to keep in mind that a service can only be removed by the service that owns it or by the "root" service.

Information about a resource can be represented in JSON, XML or HTML format, but we can only create a service by sending information with the first two ones.

Below is an example of a service described in XML format:

```
<service>
    <name>Facebook</name>
    <id>123okopwqeop21893</id>
    <key>u94832er893wjhr893j98</key>
</service>
```

We will see all this parameters in greater detail in a later section, when we talk about the security that we have implemented in this API.

As we saw before, we can retrieve information through a service with special permissions. In fact, this service is the only one allowed to manage the ownerships of the rest of services, controlling the creation and deletion of their rooms. Therefore, it is a integral part of the entire architecture. Its aim is to facilitate the work of administrators and that is why it is possible to provide information of each service in HTML format.

D. Token

The last resource of this API is the one that allow us to give access to end users. In the next section we will explain in detail its functionality, but now we are going to define the operation that can be used to create them. This operation is a little different because this is a special resource and, as we have already said thought essentially to authenticate users of other services. Besides it can be used to check that these users who are going to take part in conferences come from services that are the owners of such conferences.

The typical use case, for instance, could be one in which one of these resources takes part is the creation of one token. Tokens being nothing more than unique identifiers randomly created that have a limited time to live. In other words, some minutes later (usually three minutes), these token cannot be used and they are removed from the system. The way to create them is shown in the next example:

```
POST /room/MyRoom/token HTTP/1.1
Content-Type: application/xml
Content-Length: 63
[Security info]
[CRLF]
<token>
    <username>Bob</username>
    <role>participant</role>
</token>
```

When the server receives this request, Nuve creates a new unique identifier and relate it through a database to a username, a role and a room. Thanks to this, when a user wants to connect to one session, Nuve can retrieve this data from the database in order to know the room where the user wants to participate, with which username, and what role the user will play in the session. The response of Nuve would be the next:

```
HTTP/1.1 301 Moved Permanently
Content-Type: application/xml
Content-Length: 26
Location: /room/MyRoom/token/123p2j13io21
[CRLF]
<token>
    <username>Bob</username>
    <role>participant</role>
    <id>123p2j13io21</id>
</token>
```

It is necessary to know that each operation has the scope of the service that requests it and Nuve is aware of it thanks to the security information that the service includes in each of the requests. Also, as stated above, there is a special service called "root" that has enough permissions to operate over everything else.

We can see a summary of all operations that each service can use on each resource. The first table explains the operations that we can do with information in XML and JSON format, and the second one is for information represented with HTML.

TABLE 2
HTTP METHODS USED FOR XML/JSON CONTENTS

	GET	POST	PUT	DELETE
/user	X			X
/room	X	X	X	X
/token		X		
/service	X	X		X

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8285
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8285

TABLE 3
HTTP METHODS USED FOR HTML CONTENTS

	GET	POST	PUT	DELETE
/user	X			X
/room	X			X
/token				
/service	X			X

We can deduce the importance of the service authentication from the description that we have made through this section. Due to this, in the next one we will comment further details about it.

VI. SECURITY

This section describes the architecture of the solution implemented in order to provide the system with a security layer. The adopted mechanism is based on the combination of an extension to the HTTP header and the use of tokens for final user access.

A. Description of the problem and previous experiences

The next subsections explain the different approaches considered when designing the security solution for Nuve. Firstly, a general definition of what is considered as security in this kind of applications is given. After that, a brief analysis of the existing implementations is presented in order to provide some background on the existent works.

1) Security requirements in collaborative software

Before focusing on the problem, it is important to define the general security concerns in collaborative software (CSCW: Computer Supported Collaborative Software from now on).

A wide variety of software applications fall under the definition of CSCW, ranging from relatively simple document repositories to full blown real time multimedia systems that allow for much more complex interactions among users. The collaborative software matrix [12] perfectly illustrates that fact.

Due to that environment variety, security in this context is a little hard to define. A number of studies have been published trying to unify the definition of security requirements as well as the notation used to describe them properly ([13] and [14]). As a result of the analysis of those publications it is possible to extract a subset of rules that try to cover the most critical security problems:

- Participants follow the previously established workflow.
- The existent roles are consistent as well as the constraints imposed to them.
- Only authorized users can access the system.
- Users enter the system with the corresponding roles.
- Any temporal or conditional constraints can be applied to the resources.
- Each user's private data cannot be accessed by anyone else.

When it comes to videoconferencing (same place, different time in the matrix) the number of roles usually is very limited, simplifying interaction compared to other schemes with more complex workflows.

According to this, we reach the basic conclusion that in a context of videoconference where the usual workflow is quite simple, the main security concerns come from the authentication of the users and their authorization so they have access to the right resources depending on their roles.

2) REST Authentication

Having analyzed the security needs in a general context, it is time to study the different mechanisms that have already been implemented and published but focusing on REST which is the interface used in our system.

The most important alternatives researched and, as such, shown in this paper are: Amazon S3 [15] and OAuth [16]. Both were chosen because they are used broadly today and are proven to work.

First of all, it has to be noted that both are based, above all, on modifications of the standard HTTP authorization header defined in [17].

To sum the process up, it consists on the server responding to non authorized requests with a 401 code including a WWW-Authenticate field which specifies a challenge for the client. The next call coming from the client should include an Authorization field which is completed with the data implied by the challenge. Finally, the server should check the reply and process the request if it meets the requirements.

No specific type of authentication is enforced although in [18] two are proposed: Basic and Digest. The first one is really simple including only two fields (name and password) which are transmitted in plain text. Digest authentication is a little more complex as it proposes the use of MD5 [19] cryptographic hashing and nonce values to avoid replay attacks.

After this brief digression we switch our focus back to the studied systems. Amazon S3 is an online storage service offered by Amazon Web Serviced designed to make life easier for web applications developers with important needs in terms of data space but lacking the required infrastructure. It offers its users both SOAP and REST interfaces allowing them to perform various actions like adding or removing data. It is a paid service and, as such, it is extremely important for its users to be authenticated.

The modification of the HTTP header developed by Amazon is called "AWS". In this type of authentication, the reply to the initial 401 code is to introduce in the said header the following line: Authorization: AWS AWSAccessKeyId:Signature where AWSAccessKeyId is

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8285
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8285

given by Amazon to the client after registering as well as another key which will be used (together with a string containing several values) to calculate the signature by means of the HMAC_SHA1 algorithm.

This signature can be easily reproduced at server-side because the client's id and key are already known.

The other studied case we will explain here is the OAuth protocol which, among other security mechanisms, includes one similar to ones explained above. OAuth allows a user to grant access to their information on one site (the Service Provider), to another site (called Consumer), without sharing all of their identity. The communication between the provider and the consumer is not very different in concept to the one we see in Nuve.

The use of the HTTP header is quite similar to Amazon's but the information provided in it is not always the same and changes depending on the action requested. As a consequence, the data included in the header is not only used for authentication but also for application level purposes.

Furthermore, and without getting into too much detail, OAuth authentication uses a token which is given to the consumer to access the provider's resources. In the process, the client gives his or her credentials to the provider but never to the consumer.

To conclude, both alternatives aim to authenticate and authorize the agent that is requesting the operations. We conclude that, in the case of Nuve, this type of mechanism satisfies the security requirements obtained from the first part of this section. However, it is necessary to develop a more specific solution for our system.

B. Security in Nuve: MAuth

1) HTTP Authorization

Regarding HTTP Authorization, the path chosen is similar to the ones described above, that is, an extension to the HTTP standard header. However, like OAuth, the header is also used to carry parameters used by the application.

The full header containing all possible parameters used is as follows:

```
Authorization: MAuth
     realm="http://marte3.dit.upm.es",
     mauth_signature_method="HMAC_SHA1",
     mauth_serviceid="ServiceName",
     mauth_signature="jlsa731232=",
     mauth_timestamp="1231321321",
     mauth_cnonce="123123aadf",
     mauth_version="3.1",
     mauth_username="user",
     mauth_role="participant"
```

Below we describe individually each one of them:

- mauth_signature_method: It indicates the signature method used to sign the request. HMAC_SHA1 is the only one supported. The key used in the process is simetric and is exchanged off channel.
- mauth_serviceid: A unique identifier for the service. It is used by Nuve to obtain the key and for several other purposes at application level.
- mauth_signature: The signature generated by the method explained above.
- mauth_timestamp: Unless otherwise specified, the number of seconds since January 1, 1970 00:00:00 GMT. The value must be a positive integer and must be equal or greater than the timestamp used in previous requests.
- mauth_cnonce: A positive integer that must be different for each request with the same timestamp. Used to prevent replay attacks.
- mauth_version: Current version number.

All the parameters mentioned above are obligatory. The next two are only used for requesting access for a user to a room.

- mauth_username: Name of the user trying to access the conference.
- mauth_role: Role of the user in the conference. The possible roles a user can take in a room are: "participant", "administrator", and "observer". Each role defines limits to what a user can do while the conference is taking place.

The string used to calculate the signature varies depending on the parameters included in the header.

The format of the string is:

```
(mauth_serviceid, mauth_timestamp, mauth_cnonce,
[mauth_username, mauth_role])
```

The parameters between square brackets are only present when needed.

To better understand the flow of the authentication, let's study a particular case. A service wants to obtain the list of the existent conference rooms.

Initially, the service issues a request to Nuve:

```
GET /rooms HTTP/1.1
Host: marte3.dit.upm.es
```

The request did not include authorization information so the Nuve server replies with a 401 code indicating that the request was not authorized and providing information about the authentication type that should be used.

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: MAuth,
  realm="marte3.dit.upm.es"
```

Now, the service knows that Nuve uses MAuth to authenticate requests and must fill in every parameter to have its request approved and processed:

```
GET /rooms HTTP/1.1
WWW-Authenticate: MAuth,
  realm="marte3.dit.upm.es",
```

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8285
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8285

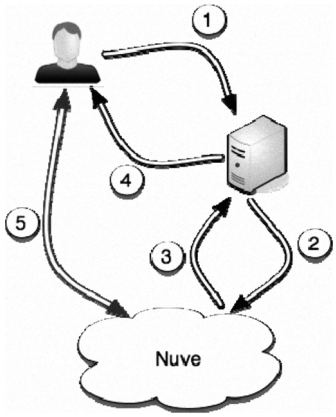

Fig. 3 Authentication messages

```
mauth_signature_method="HMAC_SHA1",
mauth_serviceid="global",
mauth_signature="dasawaraj212312",
mauth_timestamp="32132131",
mauth_cnonce="654sa5d6asdads",
mauth_version="3.1"
```

In this particular case, we are not using mauth_username and mauth_role as they are not needed for this request.

Once this point is reached, the server has all the needed data to verify the authenticity of the request, if everything is right it replies the service with a message including the list.

2) User authentication: tokens

This subsection gives a detailed explanation of the process of authenticating users in Nuve without the need of directly exchanging information between the final users' pc and the Nuve server.

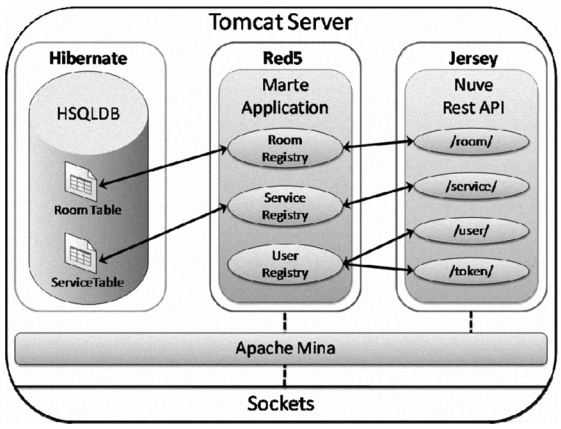

Fig. 4 Implemented code modules

This is achieved via the previously mentioned token. Only users that own a valid token have access to a Nuve conference. As explained above, tokens are generated dynamically every time a service asks for one, besides, information concerning the user (the service it belongs to and the role he or she will play) is stored in the server. Furthermore, tokens have an expiration date rendering them useless after a given period of time.

The usual workflow followed after the creation of a conference is illustrated in Fig.3.

1. The user requests access to a videoconference room. In order to do that, he or she probably has previously identified himself in the context of the service.

2. The service issues a request to Nuve asking for access to the conference for that specific user. That request includes all the authorization data mentioned before.

3. If the authentication is successful, Nuve sends a valid token back to the service.

4. The client is redirected to a dynamically generated web page which contains the Flash client needed to participate in the conference. The token contains data about the conference room and the user's role so he or she does not have to introduce any information and can only access the conference that was requested by the service.

5. The user can start interacting with others.

VII. RESULTS

This section describes the implementation of a proof of concept that we used to test the new architecture and draw conclusions about the results. Firstly, we will give a brief explanation about the test environment for this implementation detailing each of the components used and finally we will present all the tests performed and the results obtained from them.

A. Implementation

The API implementation has its roots in the Marte 3.0 application which, as we have explained before, was developed in a previous work. This application used an Apache Tomcat server on top of which a Red5 server was installed. Red5 allows for voice/video communications to be established among Adobe Flex/Flash clients. As Tomcat was an already set piece of the architecture we decided to use a Java library and we chose Jersey [20], developed by Sun and quite useful for creating RestFul APIs easily.

The working API had to embody the theories exposed throughout this article, so we started by defining the four mentioned resources (rooms, services, users and tokens). The logic behind the creation, modification, removal and reading of each of the resources was in its core a simple call to already implemented Marte classes so we did not have to rework all the parts concerning the management of the application. The most important parts reused from Marte are the following:

The RoomRegistry component takes care of all the functionality regarding creation and removal of rooms. Besides, it does so safely avoiding all the possible problems that might come up when performing those actions. For instance, when a room is deleted, RoomRegistry disconnects all the users present in those rooms providing the needed

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8285
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8285

explanation to the clients. Furthermore, it checks whether a service is authorized to delete a room.

The ServiceRegistry module is quite similar but it deals with services. When a service is created, it performs serveral actions like analyzing the existence of any conflicts with other subscribed services and when a service is deleted it deletes all its owned rooms by using the RoomRegistry component.

The last component used from Marte 3.0 is the UserRegistry which allows us to add or remove users from a room. For instance, we will use this component when a room is deleted via RoomRegistry to remove all the users from it.

Finally, to take care of all the persistence needs of the application, a HSQLDB database is used through Hibernate. It only stores data about subscribed services and rooms so two related tables were created on for each.

In Fig.5 the logic behind the removal of a service is shown as an example.

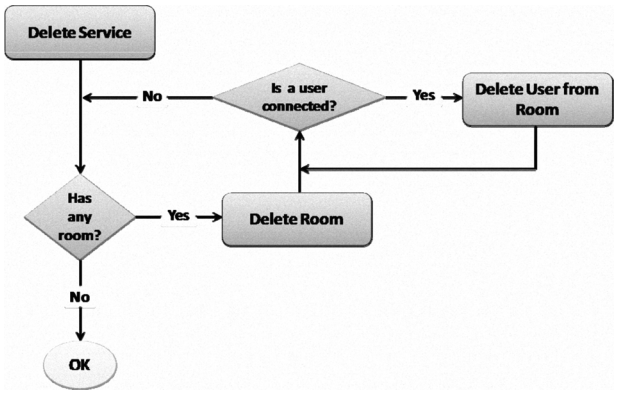

Fig. 5 Service deletion programming logic

B. Test environment

The objective of these tests was to measure the capacity of the Nuve server in terms of the number of users that was able to support, the bandwidth of a common session, monitoring the CPU usage of the machine in which the server is hosted.

The test system was a VMWare virtual machine running on top of an Intel Core 2 Duo with 2 GB of RAM. The virtual machine is limited to one CPU and 512 MB of RAM. The operating system used is an Ubuntu Linux 8.10.

In order to get this data we deployed the previous implementation in a machine to which different users from the same subnet were connected, all of this through an Ethernet connection and a Switch that supports a bandwidth of 1 Gbps. All the users and the server had network interfaces of 1 Gbps.

Regarding the bandwidth consumption we show different figures that we explain below.

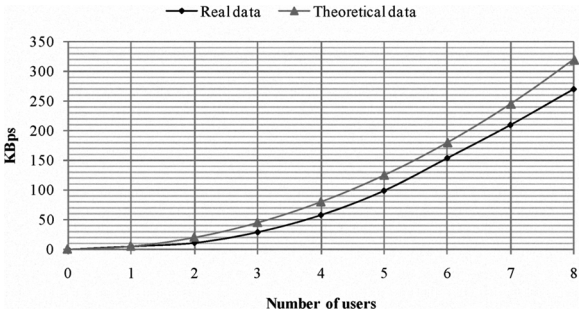

Fig. 6 Bandwidth consumed by the audio

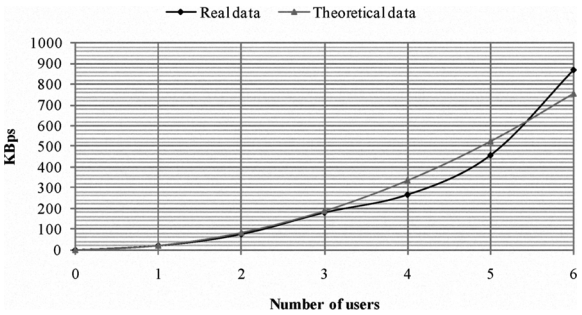

Fig. 7 Bandwidth consumed by both audio and video

At Fig. 6 we show the measured data on bandwidth that was used during a conference in which users only communicated using their voice and they did not use webcam or screen sharing. During these tests all users sent audio through their microphones at all times and simultaneously. We can also see an empirical approximation to these results that represents the next equation:

$$BW_{total} = {N_{users}}^2 \cdot BW_{audio}$$

Being BW_{total} the bandwidth consumed by the centralized server (which contains the Marte application), N_{users} is the number of users that are connected to the videoconference and BW_{audio} is the bandwidth consumed by the audio of each user (in the testing all users consume the same bandwidth).

Based on the bandwidth measured during the tests and applying the equation we get an approximated value for the bandwidth used by each user, that is 5 KBps.

Fig. 7 shows the bandwidth consumed by a session like the previous one, but in this case users are also sharing the video streams produced by their webcams. This test helps us to calculate the average bandwidth consumed by each user. We assume that the worst case is that in which all users are continuously in movement (that is the instant in which more bandwidth is going to be used). In this case the measured values should be represented by the next equation:

$$BW_{total} = {N_{users}}^2 \cdot (BW_{audio} + BW_{video})$$

We can get from this equation that the bandwidth consumed by each user that is sharing its video is 16KBps.

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8285
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8285

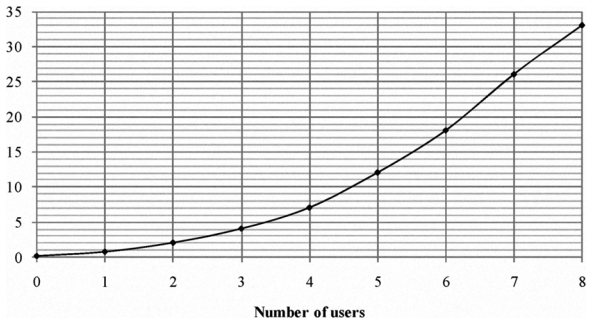

Fig. 8 CPU percentage consumed by the audio

At Fig. 8 we see how the percentage of CPU consumed varies in an audio conference, while at Fig. 9 we see the same data but in a conference that includes video and audio. As a result, we can infer that there are not many differences between videoconferences and audio conferences in terms of CPU usage in the server.

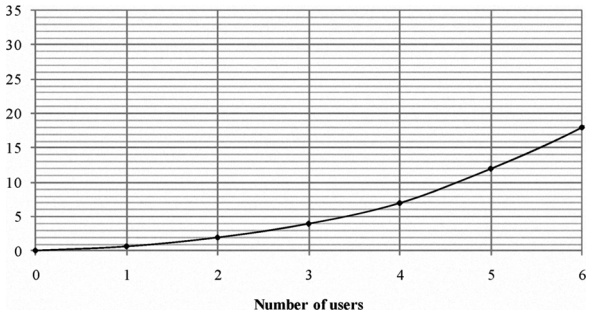

Fig. 9 CPU percentage consumed by both audio and video

It is important to notice that these tests were made in scenarios in which all users were connected to the same conference room. In a test with many rooms, the maximum bandwidth capacity of the server could be calculated by the next equation:

$$BW_{total} = \sum_{i} BW_{Room\ i} = \sum_{i} N_{users\ from\ i}{}^{2} \cdot (BW_{audio} + BW_{video})$$

VIII. Conclusions and Future Work

Throughout this paper we have shown the main features of the proposed architecture in the context of other Cloud Computing systems. We have detailed the resources oriented architecture of Nuve and as well as provided a description of the service interface. A prototype of a Nuve system has been described validated and performance measures have been provided, showing that this architecture can be easily implemented in a cost-effective way. The extension of this implementation to scalable Cloud Computing services which could provide tons of virtual rooms to users seems to be straight forward by adding a virtual room allocator among the Cloud of virtual room servers.

The most critical part of this work has focused on the development of a security mechanism which enabling the integration of the service into third parties' applications and mash-ups.

The same core system has been reused in various projects we are working on. As part of these projects we have successfully installed the core in Linux, Windows, Mac OS X and Solaris. That is, we only have to maintain one group of physical CPUs which, in turn, have virtual machines running on top. The different projects are represented as services and they can all share the same, unmodified, core. Besides, while currently new virtual machines have to be set up by hand, we are working on the automation of the process. As a showcase we have integrated Nuve into Google Wave [21].

The Flex/Flash Rich Internet Applications (RIA) development framework from Adobe has proved to be very effective for implementing the prototype. The old Marte 3.0 client-server system was transformed into the Nuve architecture in three months by two persons working 60% of their time on it which seems to be very reasonable resource expenditure for such a development.

Using the Flex/Flash RIA based videoconferencing has the additional benefit of avoiding the user to have to install any application in most cases because most browsers today have the Flash Player plug-in installed.

Finally, even though the tests are focused on measuring the limits of the system and do not represent a real scenario where usually one user sends more information than the others, they serve us to estimate the maximum video and audio bandwidth consumption by a normal user in this first implementation. Regarding server CPU usage, the results show that in future works we have to design a low-level architecture that can be scaled through several server machines without overloading any of them. In order to achieve this scalability and guarantee some QoS, we will need to instantiate virtual machines and turn then on and off. This motivates us to follow the Cloud Computing principles.

References

[1] Google Video Chat. [URL] http://mail.google.com/videochat

[2] Skype. [URL] http://www.skype.com/

[3] Ribbit. [URL] http://www.ribbit.com/

[4] ooVoo. [URL] http://www.oovoo.com

[5] J. Cerviño, P. Rodríguez, J. Salvachúa, G. Huecas y F. Escribano, "Marte 3.0: una videoconferencia 2.0" JITEL 2008, pps: 209-216 16-18, September 2008.

[6] P. Mell and T. Grance, "Draft NIST Working Definition of Cloud Computing", January 2009.

[7] L. Wang, G. Von Laszewski, M. Kunze and J. Tao, "Cloud Computing: a Perspective Study", Dec. 2008.

[8] Amazon Elastic Compute Cloud (Amazon EC2) [URL] http://aws.amazon.com/ec2/, access on June 2009.

[9] Flash Player Penetration [URL] http://www.adobe.com/products/player_census/flashplayer/, access on June 2009.

[10] JSON [URL] http://www.json.org/, access on June 2009

[11] Cesare Pautasso and Erik Wilde, "From SOA to REST - Designing and Implementing RESTful Services". Tutorial at 18th Int. World Wide Web

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8285
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8285

Conference, Madrid 2009. [URL] http://www2009.org/tutorials/T9-F.html, access on June 2009.

[12] R. Johansen, "Groupware: Computer support for business teams". New York: The Free Press 1988.

[13] Tanvir Ahmed, Anand R. Tripathi, "Static Verification of Security Requirements in Role Based CSCW Systems", Symposium on Access Control Models and Technologies, 196-203 Como, 2003. ISBN: 1-58113-681-1.

[14] A. Tripathi, T. Ahmed, and R. Kumar. "Specification of Secure Distributed Collaboration Systems. IEEE International Symposium on Autonomous Distributed Systems (ISADS), April 2003.

[15] Amazon S3 [URL] http://aws.amazon.com/s3/, access on June 2009

[16] OAuth [URL] http://oauth.net/, access on June 2009.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frysyk, L. Masinter, P. Leach and T. Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1", RFC 2616, June 1999.

[18] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, E. Sink and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[19] R. Rivest, "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[20] JSR 311: JAX-RS: The Java API for RESTful Web Services [URL] http://jcp.org/en/jsr/detail?id=311, access on June 2009.

[21] Google Wave. [URL] http://wave.google.com

Digital Object Identifier: 10.4108/ICST.COLLABORATECOM2009.8285
http://dx.doi.org/10.4108/ICST.COLLABORATECOM2009.8285