Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Informàtica

Títol: Estudi i disseny d'un sistema de comunicació directa amb WebRTC .

Document: Memòria

Alumne: Andreu Mañosa Crous

Tutor: Jose Luís Marzo Lázaro

Departament: Arquitectura i tecnologia de computadors

Àrea: Arquitectura i tecnologia de computadors

Convocatòria (mes/any): Setembre 2016

Índex de continguts

1.Introducció

L'objectiu principal del treball és dissenyar i implementar un sistema de comunicació directe per una aplicació Web, que permeti comunicacions de text, veu i vídeo entre diversos usuaris, usant l'arquitectura de comunicació d'igual a igual (P2P) i les API's de *JavaScript* de *WebRTC*, *Socket.IO i AngularJS*.

El sistema gestiona el registre i autentificació pels usuaris, la seva llista de contactes, les converses entre usuaris autentificats i permet establir videoconferències amb vídeo i/o àudio entre ells.

El projecte podria separar-se en 2 parts, la part per construir tot el codi referent a la interacció amb l'usuari i una part per a la construcció d'una API que faci de pont entre la part per a la programació de videoconferències en WebRTC i la part referent a controladors i interfícies d'usuari.

L'ús de la tecnologia de WebRTC per a les videoconferències, llançada per Google l'any 2011 permet la comunicació d'igual a igual (entre navegadors) a temps real sense la necessitat de la instal·lació de "plug-ins". Però **la multi comunicació entre més d'un navegador (usuari) fa que haguem de tenir tantes comunicacions P2P com usuaris estiguem parlant**, cosa que fa evident un problema d'escalabilitat entre converses de més d'un usuari (o múltiples converses per part d'un mateix usuari), degut a que cada canal d'intercanvi de dades que s'estableix entre dos usuaris fa acostar cada cop més als dos al límit de l'ample de banda màxim del que disposen.

Per solucionar això, **WebRTC permet regular la velocitat de transmissió (VDT) que consumeix cada canal P2P que s'estableix entre dos usuaris**, des de un mínim de 60 kbits/s de VDT, fins a la màxima taxa de bits que es pot transmetre per unitat de temps (depèn del que utilitzem per a connectar-nos amb el DSLAM de la companyia (ADSL, Fibra...)).

Per regular aquesta VDT, WebRTC dona l'opció de modificar els paràmetres d'inicialització dels fluxos multimèdia que s'envien. Malgrat ser un estàndard de W3C actualment no està suportat pel navegador Firefox.

D'aquesta manera l'aplicació busca poder oferir conferències entre els usuaris amb una qualitat (qualitat d'imatge i so) adaptable dinàmicament a les característiques de xarxa de cada usuari.

1.1. Motivació i propòsits

Motivacions

La principal motivació que m'ha portat a fer aquest treball ha estat la possibilitat d'estudiar i poder conèixer millor la tecnologia de comunicació *WebRTC*, ja que és una tecnologia bastant nova i està sent usada per moltes de les marques capdavanteres en aplicacions informàtiques. Apart d'això, també em motivava poder estructurar per primer cop un servidor Web des de zero, amb la instal·lació dels servidors Apache i Node.js (juntament amb els mòduls necessaris per a complementar-los) i un sistema de gestió de base de dades.

A més del sistema de multi conferències també m'ha interessat molt realitzar un sistema amb control d'usuaris, proporcionant-los-hi una interfície per la gestió de contactes, control de converses, i la possibilitat d'intercanviar-se de manera persistent tot tipus de continguts.

Actualment hi han varies plataformes que proporcionen API's per a la realització de videoconferències en *WebRTC*, però al estar *WebRTC* encara en desenvolupament per tots els navegadors el programador queda molt lligat a que la API que utilitza actualitzi les llibreries d'acord el suport que va donant *WebRTC* per cada navegador. Per això en el treball es desenvolupa una pròpia plataforma independent de cap altre API.

Raons que justificaven el desenvolupament

Aquest treball apareix per eliminar una de les mancances que té el projecte VITAM, un dels projectes desenvolupats al BCDS. El VITAM (Videoconference Teleassistance and Monitoring) és una altre aplicació únicament de multi conferències Web basada en WebRTC i construïda sobre la plataforma Licode.

L'ús de la plataforma Licode a nivell de servidor tenia com objectiu solucionar el problema d'escalabilitat que ens proporcionen els múltiples canals P2P d'intercanvi de dades que s'han de tenir oberts alhora de fer una multi conferència, ja que proporcionava una arquitectura de gestió de fluxos amb un conjunt d'interfícies que permetia la necessitat d'un únic canal per a cada usuari d'una multi conferència.

Malgrat la bona arquitectura que presenta Licode es traslladen els problemes d'escalabilitat dels navegadors al servidor, fent necessària una xarxa de servidors per a poder gestionar els fluxos dels múltiples usuaris que estan en línia (independentment de la conferència en la que es trobin). A part d'aquest problema, tot i que la plataforma oferís també un sistema de gestió de fluxos a nivell de P2P, el VITAM estava molt lligat als possibles errors i actualitzacions d'aquesta plataforma que encara està en desenvolupament.

Per això, el principal propòsit del treball es muntar una arquitectura que sobrecarregui al mínim el servidor principal amb l'establiment de connexions directes entre usuaris, afegint però un alt control de l'ample de banda que ocupen aquestes connexions per tal de mitigar els afectes d'escalabilitat que es poden produir a la part client.

El treball s'ha realitzat tot de nou sense tenir en compte el que hi havia anteriorment a VITAM, degut al fet de que VITAM usava la gestió de fluxos P2P de *Licode*, i per poder tenir un sistema totalment independent era necessari fer un propi sistema de gestió de comunicacions P2P.

Que s'esperava obtenir

El que s'esperava obtenir era una aplicació Web amb una primera plana per al registre i autentificació d'usuaris. A partir d'aquí el client havia d'accedir al seu perfil, on se li oferiria la possibilitat de buscar contactes i agregar-los. A part d'això havia de tenir també la possibilitat de crear contactes múltiples.

Un cop feta aquesta part molt semblant a l'antic *Messenger*, s'havia de proporcionar la possibilitat de que els usuaris d'una conversa poguessin realitzar videoconferències entre tots ells, cosa que implicava l'ús d'una tecnologia que permetés el intercanvi de múltiples fluxos d'àudio i vídeo.

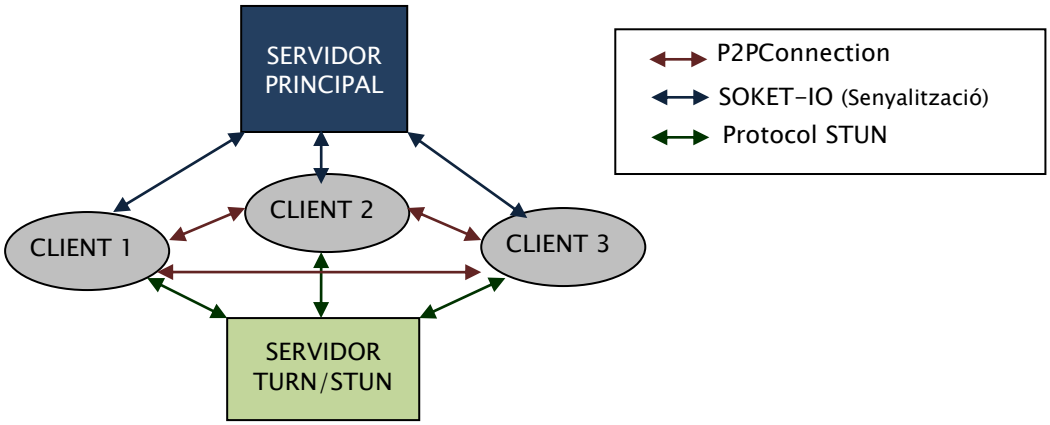
Centrant-nos en la part de les multi conferències se sabia originàriament que s'usaria la tecnologia WebRTC i es tenia la idea de regular els amples de banda per tal de poder fer escalable el sistema als usuaris, a partir d'aquí s'esperava poder oferir una interfície a l'usuari

que li proporciones les funcionalitats de comunicar-se via vídeo, àudio i xat, amés de que tingués el més alt control possible sobre aquestes funcionalitats.

1.2. Objectiu del projecte

Fites especifiques a assolir cap el propòsit general.

Per tal de complir amb l'objectiu principal del projecte han estat necessaris 2 servidors, el servidor principal (*Signaling and Controller Server*) (SCS) en el que hi ha tota el que es necessita per l'aplicació Web, i el servidor on hi han els servidors STUN i TURN. Aquest segon servidor necessita 2 adreces IP (per al funcionament del protocol STUN) i és el que permet l'establiment de connexions *WebRTC* entre clients que no es troben en xarxes públiques.



Així doncs el primer dels objectius és la construcció d'aquesta arquitectura. Pel que el servidor principal respecta s'ha hagut de preparar des de zero, però pels servidors TURN i STUN s'ha usat ja el muntatge previ que hi havia pel VITAM.

A partir d'aquesta estructura bàsica els objectius del projecte són els següents:

-Estudiar i conèixer de les tecnologies a usar, en especial *WebRTC* i el protocol SDP (*Session Description Protocol*) encarregat de limitar les velocitats de transmissió que consumeixen els diferents fluxos.

- Estudiar i conèixer els entorns de treball *AngularJS* i *JQuery* per el seu us en *JavaScript*.

-Implementar un sistema de multi conferències en temps real de manera bàsica (aquest sistema només estava pensat per la recuperació i reenviament de fluxos de vídeo de manera local).

-Estudiar les llibreries necessàries per establir un protocol de senyalització previ per poder comunicar usuaris de diferents hosts a través del servidor. La necessitat de que el canal entre clients i servidor sigui bidireccional ha portat a l'ús de *WebSockets* implementat amb la llibreria de *Socket.IO* de *JavaScript*.

-Dissenyar i programar la interfície Web del client, juntament amb la part back-end de comunicació entre clients i servidor. Per comunicar la interfície Web amb el la base de dades s'ha fet ús de la llibreria *AngularJS*, i per a fer comunicacions persistents amb el servidor per enviar dades entre usuaris s'ha usat *WebSockets*.

-Dissenyar i programar la part de gestió de la base de dades. La base de dades *MySQL* es pot gestionar manualment amb *phpMyAdmin*, però l'aplicació interactua amb ella a través dels mòduls *express* i *mysql* afegits al servidor *Node.JS*.

-Implementar el sistema de multi conferencies final relacionat amb les converses entre els diferents usuaris registrats que faci una gestió de l'ample de banda usat tenint en compte les capacitats de xarxa d'aquests.

Aquests són els objectius del projecte en els que s'espera acabar d'aprendre i posar en pràctica els fonaments adquirits en les diferents assignatures de la carrera. Pel que fa les llibreries i tecnologies mencionades en el llistat d'objectius generals només coneixia el llenguatge SQL com a SGBD relacional, així doncs, tot i que anteriorment també havia realitzat algun treball informatiu de WebRTC aprendre més a fons les tecnologies, llenguatges i llibreries usades ha estat també un objectiu secundari del projecte.

En els primers apartats de la memòria es mostren els aspectes més centrats en la viabilitat i planificació del projecte, i a partir de l'apartat 5 s'entra a explicar de manera més detallada com s'ha construït l'aplicació Web juntament amb l' explicació de les eines usades i el perquè de que s'han escollit.

1.3. Vocabulari

Durant el projecte, per abreujar, s'usen els següent acrònims:

- ASSIO (*API sobre Socket.IO*)

  Conjunt de classes destinades a la connexió del client amb els servidor a través de *sockets*.

- ASWRTC (*API sobre WebRTC*)

  Per a disminuir l'acoblament entre classes s'han implementat un conjunt de classes que gestionen la videoconferència que actuen com una plataforma, a aquesta part se l'anomena abreviadament ASWRTC.

- HTTP (Hypertext Transfer Protocol)

  Protocol que permet la comunicació entre client i servidor.

- HTTS (Hypertext Transfer Protocol Secure)

  Protocol que permet la comunicació entre client i servidor de manera segura gràcies als certificats SSL i l'ús del protocol TLS en capes inferiors.

- NAT (*Network Address Translation*)

  Procés pel qual es modifiquen @IP i ports d'àmbit local a @IP i ports d'àmbit global.

- SCS (*Signaling Controller Server*)

  Servidor de l'aplicació Web que a través de diferents ports proporciona al client Web totes les funcionalitats que necessita.

- SDP (*Session desription protocol*)

  Protocol per descriure els paràmetres d'inicialització dels fluxos multimèdia usats en un canal *WebRTC*.

- SRTM (*Secure real time multiconference*)

  És el nom de la plana Web.

- STUN (*Session Traversal Utilities for NAT*)

  Protocol usat per WebRTC per el descobriment d'adreces IP i ports públics del NAT, juntament amb les característiques del tipus de NAT que emmascara l'usuari.

- TLS (*Transport Layer Security*)

  Protocol sobre TCP ubicat a la capa de transport que permet l'enviament de dades de manera xifrada.

- TURN (*Traversal Using Relays around NAT*)

  Protocol que permet la redirecció de canals P2P impossibles d'establir entre dos usuaris a través d'un servidor de reenviament que permet que tots dos usuaris puguin iniciar l'intercanvi de fluxos.

- VDT (*Velocitat de transmissió*)

  En la memòria s'usa tant la paraula VDT com ample de banda de manera indiferent, encara que no siguin exactament el mateix en el document s'usen com a sinònims. La VDT és la quantitat d'informació en bits per segon que podem enviar i rebre a través de la xarxa.

- WebRTC (Web *real time Communications*)

  Tecnologia que permet l'intercanvi de grans quantitats de dades de manera directe entre navegadors.

2. Estudi de viabilitat

PARÀMETRES NECESSARIS PEL DESENVOLUPAMENT DEL PROJECTE

Per a poder començar a desenvolupar el projecte cal un ordinador amb el sistema operatiu Ubuntu (per la facilitat que proporciona el instal·lar mòduls i llibreries), ja que tot es pot fer agafant l'adreça d'àmbit local 127.0.0.1 (localhost) i obrint els ports necessaris per al funcionament de l'aplicació.

Per tal de poder donar servei als usuaris connectats d'arreu del món cal però una @IP publica juntament amb un mínim de 4 ports oberts a Internet, que són els corresponents per defecte a les aplicacions de xarxa *Secure SHell*, *HTTP*, *NodeJS* i *DNS*.

A més però, l'ús de WebRTC implica l'obertura d'una @IP i tres ports més. La necessitat d'un dels tres ports és degut a que el navegador Chrome no permet l'ús de la tecnologia WebRTC sinó es fa servir HTTP segur per demanar la pàgina en la que s'usa la llibreria WebRTC.

A part de la necessitat d'usar *HTTPS* per Chrome, cosa que implica un port més als quatre inicials, quan dos usuaris volen establir una conversa d'igual a igual entre ells usant WebRTC, és necessari el intercanvi de missatges del protocol d'aplicació STUN per a saber on es troben, cosa que implica l'ús d'un servidor STUN per resoldre casos en que els dos usuaris que volen establir un intercanvi de flux no ho poden fer per restriccions en les seves xarxes, ja sigui perquè estan darrera de routers amb NAT, Firewalls o altres. El servidor STUN en aquest cas necessita dues adreces IP i dos ports més, que sumats als quatre que teníem donen un total de set.

D'aquesta manera es necessiten un servidor que tingui disponibles dos adreces IP públiques i set ports oberts a Internet com a paràmetre necessari per poder començar a iniciar el projecte.

COST I VIABILITAT DEL PROJECTE

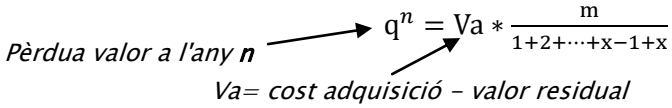
Un cop explicat el necessari per a poder començar el projecte, es pot fer un anàlisi del cost que implicaria a una empresa desenvolupar-lo de manera independent. En el meu cas, el grup de recerca BCDS m'han proporcionat el material necessari per al desenvolupament del treball, així que el cost del material i desenvolupament ha estat zero, en el cas d'una empresa però podríem pensar en la següent compatibilitat de costos:

Usant el sistema de full-costing, suposant que el projecte comença al Març i acaba a final d'Agost, cal tenir en compte les següents despeses:

Costos de desenvolupament:

Personal: És necessari un desenvolupador/programador que dugui a terme la feina. El desenvolupador ha d'estar 6 mesos treballant dedicant 4 hores cada dia excepte festius. Cosa que implica un salari total en brut de 5760€ suposant que se li paga a 12€ l'hora.

Ordinador per al desenvolupament: Considerant que el hardware en informàtica s'amortitza en x=3 anys, i es proporciona l'ordinador a l'empleat als 2 anys de la compra. Si tenim en compte que el valor residual de l'ordinador després de la seva vida útil serà de 0€, si apliquem una amortització accelerada el valor de l'ordinador al final del 2n any serà *valor=cost adquisició-*q_1*-*q_2, on q_n s'ha calculat amb la seguen formula:



D'aquesta manera el cost de l'ordinador al moment al que es proporciona al treballador és de 600€ – 300€ – 200€ = 100€.

Costos d'estructura assignables

Un cop desenvolupada o durant el desenvolupament de l'aplicació cal contractar un servidor al núvol per a tenir l'aplicació visible a Internet, més un programador que la mantingui.

En aquest cas els costos de manteniment no s'inclouen degut a que es una necessitat posterior a la presentació, però si que seria necessari afegir com a cost assignable el fet de tenir un servidor al núvol en l'etapa de desenvolupament.

En aquest cas, necessitem contractar 2 noms DNS. El preu que això suposa és:

Nom (unitats)	Quantitat	Cost mensual (€)
Voltatge nucli	10	95
RAM (GB)	12	60
Espai de disc (GB)	500	52.5
IP Pública	2	6.5

El cost mensual són 214€, que multiplicats pels 4 mesos de desenvolupament sumen 856€ als costos d'estructura.

La taula de costos final, tenint en compte els 2 costos calculats, és la següent:

Concepte	Total	
Costos de fabricació directes	5760€	} Despeses de desenvolupament
Costos de fabricació indirectes	100€	
Costos d'estructura assignables	856€	} Despeses d'estructura

D'aquesta manera **el cost total del projecte, és de 6716 €.**

3.Metodologia

Per a la construcció del sistema s'ha aplicat un cicle de vida en cascada, per tal de complir com més estrictament millor l'idea de negoci presentada. Així doncs, tenint en compte que durant el treball podien sorgir problemes en el desenvolupament de requeriments dels qual tenia un alt grau de desconeixement de la manera d'implementar-los, s'ha intentat aprofitar la rigidesa de la metodologia com a arma per l'obligatorietat en el compliment dels requisits.

Tot i que el resultat final està obert a modificacions i afegiment de noves funcionalitats, a aquest primer desenvolupament de les bases, crec que la metodologia que més se li adequava era l'escollida, ja que durant el projecte no hi havia d'haver tracte amb clients (cosa que descartava l'ús de metodologies àgils) i amés la documentació que s'havia de fer sobre el sistema havia de ser bastant amplia.

Dintre de la fase d'implementació si que s'ha seguit una estructura més semblant a la SCRUM, agafant les funcionalitats inicials de la metodologia en cascada com si fos un "product backlog", i començant per les funcionalitats més bàsiques, s'han anat afegint noves funcionalitats al sistema de manera iterativa seleccionant una funcionalitat per a cada nova iteració.

D'aquesta manera es pot esquematitzar la metodologia del software seguida tal com es mostra a la *figura 1*:

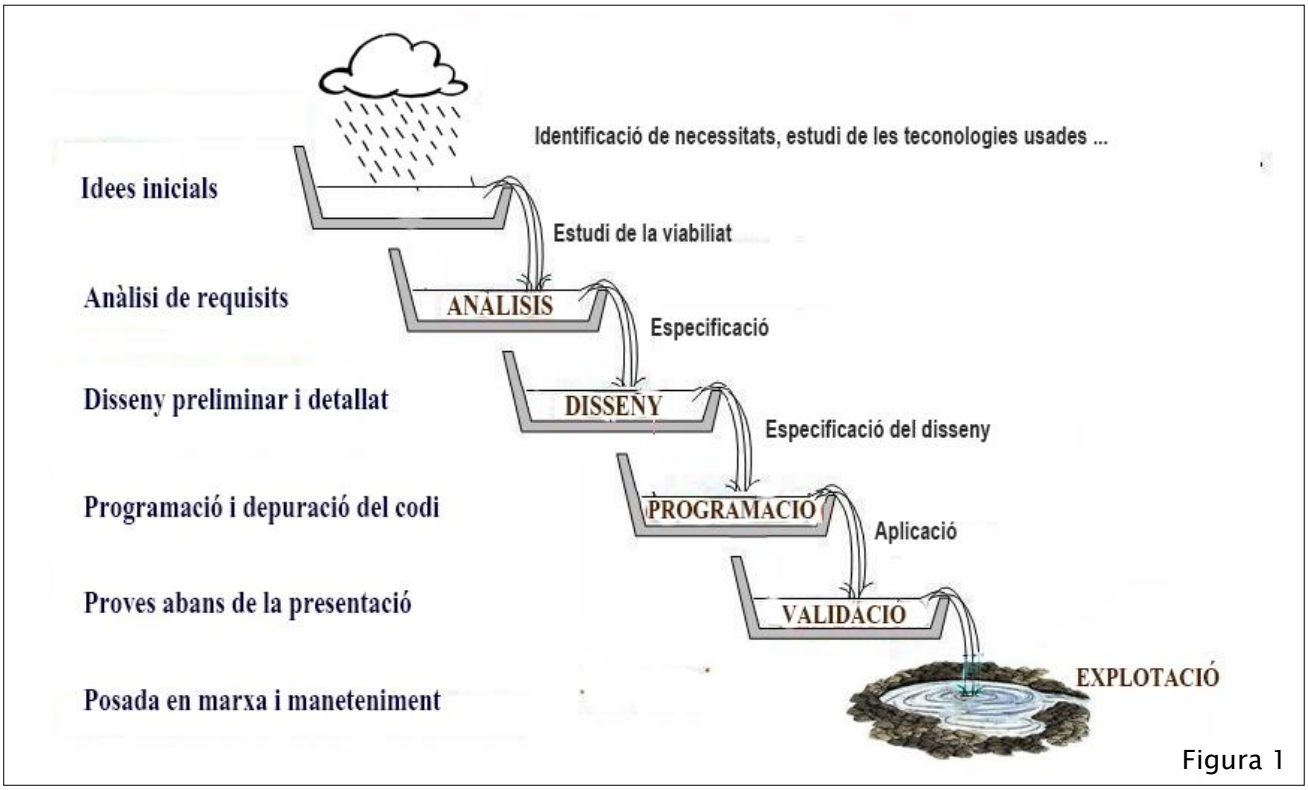


Figura 1

Un cop feta la primera presentació de l'aplicació Web crec que el més encertat podria ser seguir la metodologia SCRUM per a la gestió del projecte, tant per al manteniment com la implementació de noves funcionalitats.

4. Planificació

La planificació que es va proposar en un primer moment va ser la següent:

Tasques		*Duració*	*Data*
DOCUMENTACIÓ	-ESTUDI I PLANTEGAMENT INICIAL	30 dies	Març
	- Estudi de la tecnologia WebRTC	15 dies	
	- Estudi de la llibreria AngularJS	5 dies	
	- Estudi de Node.js	5 dies	
	- Estudi de Socket.io	5 dies	
	-ANÀLISI DELS REQUERIMENTS	5 dies	
	-DISSENY PRELIMINAR I DETALLAT	5 dies	Abril
	-PROGRAMACIÓ I DEPURACIÓ DEL CODI	90 dies	
	- Preparació del servidor	5 dies	
	- Gestió de converses	55 dies	
	- Construcció base servidor	2 dies	Maig
	- Construcció interfície Web	5 dies	
	- Comunicació amb xat	4 dies	
	- Comunicació amb vídeo	2 dies	
	- Comunicació amb xat, àudio i/o vídeo	2 dies	
	- Control estàtic de velocitats de transmissió	10 dies	Juny
	- Automatització dels ***kbps*** enviats i rebuts	20 dies	
	- Gravació de fluxos	10 dies	
	- Gestió persistent	35 dies	
	- Construcció de d'interfície de la BD	10 dies	Juliol
	- Control d'usuaris	5 dies	
	- Control de trucades	5 dies	
	- Control de missatges	5 dies	Agost
	-PROBES ABANS DE LA PRESENTACIÓ	2 dies	
	-POSADA EN MARXA I MANTENIMENT	-	Sept.

Durant el desenvolupament del projecte i han hagut petites variacions especialment degudes a la redacció de la documentació. I s'han hagut d'avançar petites parts de funcionalitats posteriors per tal de poder escriure la memòria.

Un altre problema en el compliment de la planificació ha estat el posar en pràctica per primer cop les tecnologies i llibreries usades, cosa que ha fet que en algunes parts s'agüés de dedicar més temps que d'altres degut a errors sorgits degut a la falta de pràctica.

5.Marc de treball i conceptes previs

En aquest apartat es fa primer una introducció a les tecnologies usades i llavors s'entra a especificar a on i perquè s'han usat.

5.1. Introducció a les tecnologies utilitzades

Les dos tecnologies usades són principalment les de *WebRTC* i *Websockets*. De fet tot gira entorn a *WebRTC*, ja que l'ús de *WebSockets* va ser degut a la necessitat d'establir canals bidireccionals entre servidor i clients alhora de fer procés de senyalització.

Per altre banda, per tal d'utilitzar la llibreria *Socket.IO* per la programació dels *WebSockets* tant en la part servidora com client és necessari usar un servidor que es pugui implementar en *JavaScript*, i per això es va usar l'entorn de programació *NodeJS*.

Dit això ara s'introduirà per separat cada una de les tres tecnologies, donant major importància a la part de WebRTC. Apart de les tecnologies s'explicarà que és exactament AngularJS.

5.1.1.WebRTC

Web Real-Time Communication (WebRTC) és un estàndard que estén el model de navegació web, donant la capacitat als navegadors de poder intercanviar fluxos de dades entre ells de manera directe (p2p) sense la necessitat d'usar "plugins".

L'any 2011 WebRTC va ser llançat com a un projecte open source per Google i posteriorment fou estandaritzat pel *World Wide Web Consortium* (W3C) com a API de JavaScript.

L'API WebRTC gira sobre els conceptes de ***Mediastream***, ***PeerConnection*** i ***Datachannel***, proporcionant un conjunt de mètodes per cada un dels 3 blocs. Tot seguit s'especificarà una mica cada part, però abans, és mostra l'**arquitectura bàsica de WebRTC**.

ARQUITECTURA DE WEBRTC

L'arquitectura que s'ha usat, que és la que està pensada per l'ús de WebRTC, és la següent:

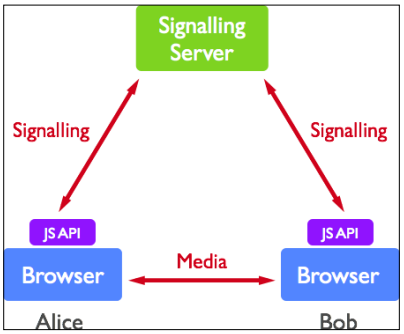


Figura 2

Com es pot veure a la *figura 2*, el flux multimèdia va directe entre navegadors, però per tal d'establir un primer contacte, és necessari un servidor de senyalització que "connecti" els dos usuaris que usen el navegador. El diàleg entre el servidor i els navegadors és l'anomena't protocol de Signaling, que no necessita més que enviar d'un navegador a l'altre les *descripcions de sessió* fetes amb el protocol SDP, i que s'han d'intercanviar els usuaris WebRTC per poder començar.

Gràcies a la interfície de programació que WebRTC proporciona per a Javascript podem estalviar-nos molta feina al programar un sistema de videoconferència, anem a veure ara els mètodes més importants de l'API.

APIS DE WEBRTC

MEDIASTREAM

És una representació abstracta d'un flux de dades, àudio i/o vídeo. **És com un controlador per a la gestió dels fluxos; permetent mostrar el contingut dels fluxos, gravar-los, o enviar-los a traves d'un "peer" remot (flux que va cap a fora)**.

Un MEDIASTREAM pot tenir zero o múltiples pistes (tracks) que es sincronitzen per a ser representades. Cada ***track*** es un objecte ***MediaStreamTrack*** que representa un tipus de medi diferent (àudio o vídeo) i alhora, un *MediaStreamTrack* pot estar compost per un o més canals amb els que capta la informació (per exemple la pista d'àudio pot tenir 2 canals, l'altaveu esquerra i el dret).

Es pot representar esquemàticament un objecte MediaStream tal com es mostra a la *Figura 3*:

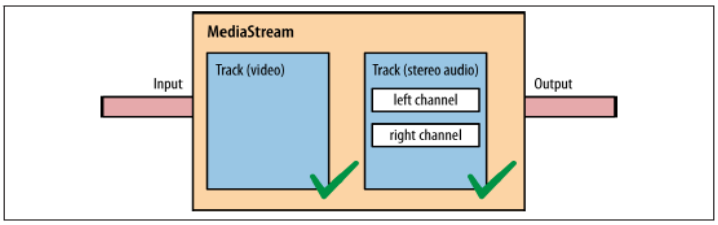


Figura 3

Els principals mètodes que proporciona la llibreria *MediaCapture* de W3C per operar amb un objecte MediaStream són els següents:

- ***getUserMedia(restriccions,successCallback,errorCallback)***

  Permet obtenir accés a un dispositiu local d'entrada d'àudio i/o vídeo, especificant un conjunt de restriccions (obligatòries o opcionals) i 2 mètodes dels quals un s'executarà de manera asíncrona depenent de si s'ha obtingut l'accés al dispositiu o no. El getUserMedia permet afegir dinamisme als tags *<img>*, *<video>* i *<audio>* de *html*, podent-los-hi associar el flux de dades recuperat perquè el mostrin.

- ***creatreObjectURL(stream)***

  Fa que el navegador creï una URL única per un fitxer local o un objecte binari (blob). Per exemple podem crear una blob URL per un objecte MediaStream. Aquesta URL es podrà usar per referenciar els MediaStream locals o remots dins de la pàgina HTML.

- ***stop()***

  Esborra l'accés al *LocalMediaStream*.

Altres mètodes de l'API i el seu ús descrit en el següent enllaç [1]

D'aquesta manera la **llibreria MediaStream ens permet obtenir i transmetre fluxos de dades de dispositius com càmeres, microfons o altaveus i intercanviarlos amb altres usuaris.**

PEERCONNECTION [2]

Una connexió entre dos "peers" (PeerConnection) és el canal que permet a dos usuaris comunicar-se directament a través dels seus navegadors, i es gestiona també a través de les llibreries que WebRTC proporciona.

Quan volem establir una línia d'intercanvi de flux amb un altre usuari es necessari crear un objecte *PeerConnection* que ens ho permetrà. Quan es crea l'objecte se li associa automàticament un ***Interactive Connectivity Establishment Agent*** (ICE Agent) al que opcionalment se li passa la direcció d'un servidor STUN i TURN. Un cop creat l'objecte, tant a la part local com a la remota es necessari, abans de tenir el canal operatiu, un intercanvi (entre "peers") dels possibles canals pels quals es poden enviar dades, i les restriccions de la sessió de cada usuari.

Del l'obtenció de canals candidats se'n encarrega l'ICE agent al complet, i per la creació de les restriccions de sessió, la mateixa classe *PeerConnection* proporciona els mètodes ***createOffer()*** i ***createAnswer()***, tot i que l'ICE Agent també en controla el procediment. En els dos casos però, per poder fer arribar les dades creades d'un usuari a un altre, cal implementar un **protocol de senyalització** apart amb canals bidireccionals que passin per un servidor.

D'aquesta manera, només en la creació del canal entre els dos "peers" és quan WebRTC necessita l'ajuda d'un servidor entre usuaris. La resta d'intercanvi de dades és fa d'igual a igual.

Centrant-nos en **l'ICE Agent, cal dir que és l'aspecte més important en una connexió WebRTC, i s'encarrega de:**

- Obtenir les *tuples* (IP adress, port) candidates per les que es poden enviar i rebre els fluxos. Busca una o més *tuples* candidates per a cada objecte MediaStreamTrack i cada cop que troba un nou candidat ho notifica al "peer" a través de l'event ***onicecandidate*** de la classe *PeerConnection* i que per tant cal implementar.

  Bàsicament el que es fa en el mètode que es crida després de l'event ***onicecandidate*** és enviar el candidat per afegir-lo al *PeerConection* de l'usuari remot a través del mètode ***addIceCandidate***.

- Realitza el control de l'estat de connectivitat entre els "peers". L'ICE Agent d'un "peer" inicialment comença amb l'estat de búsqueda (*gathering*) i a partir d'aquí, quan rep candidats ICE de l'altre "peer" el que fa és bàsicament comprovar totes les combinacions entre els seus candidats i els candidats remots, enviant-se missatges del protocol STUN amb l'altre ICE Agent. La proba de canals la fan de manera ordenada segons la complexitat de la connexió, és a dir, comencen pels candidats que a priori aniran millor i van anant a pitjor. Quan troben almenys un parell de candidats que funcionen per cada *MediaStreamTrack* l'estat passa a ser *completed*, o *failed* si no s'ha troba cap canal entre els usuaris.

- Envia senyals per mantenir el canal obert (*Connection keepalives*)

Al igual que el intercanvi de candidats té associat els events *onicecandidate* i *addIceCandidate*, el intercanvi de la **restricció de sessió** entre usuaris és fan a través dels mètodes ***createOffer()***, ***createAnswer()***, ***setLocalDescription()*** i ***setRemoteDescription()***. Al crear, un Offer o Answer, passant-li com a arguments les restriccions que vulguem, es crea un objecte SDP amb el qual

podem cridar el mètode *setLocalDescription(* SDP creat *)* que s'encarregarà de posar la restricció del nostre objecte PeerConnection, un cop fet això podem enviar l'objecte SDP creat a l'altre usuari perquè sàpiga quines restriccions li posem sobre el seu flux de dades que ens vol enviar.

Com a últim detall del PeerConnection cal dir que, al ser un canal P2P entre els 2 usuaris, WebRTC presenta un problema. Alhora de trobar els camins candidats pels quals s'establirà el canal, **si dos usuaris estan en xarxes privades (o encara que sigui un només) l'ICE Agnet del o els usuaris que estan a la xarxa privada no sabrà cap tupla (@IP, port) que li serveixi a l'altre "peer"**. Però l'ICE Agent té la capacitat de **solucionar-ho si se li proporciona la direcció d'un servidor STUN en el millor dels casos, i un servidor TURN pels cas en que tots dos usuaris estan en xarxes privades**.

DATACHANNEL

El tercer gran bloc que la llibreria de WebRTC proporciona és la possibilitat de tenir un canal de dades. El canal de dades es pot incorporar al l'objecte PeerConnection i per tant, per l'objecte PeerConnection podem enviar i rebre els objectes MediaStream local i remot, més els objectes DataChannel tan local com remot.

La creació del DataChannel només implica la implementació de l'event ***ondataChannel*** de la classe *PeerConnection* que es crida quan l'usuari remot ha cridat el mètode ***createDataChanel***, també de la classe *PeerConnection*.

Un cop fet això només cal implementar els events ***onmessage*** (per a mostrar els missatges rebuts), ***onopen*** (es crida un cop podem enviar missatges) i ***onclose*** (per tancar el canal de dades).

Vistos els tres grans apartats en que es divideixen les API's de WebRTC ara es mostrarà el procés bàsic de WebRTC per tal de traçar una PeerConnection entre dos usuaris per intercanviar fluxos multimèdia.

PROCEDIMEN BÀSIC DE WEBRTC

A partir d'aquesta base s'ha construït la part WebRTC del projecte:

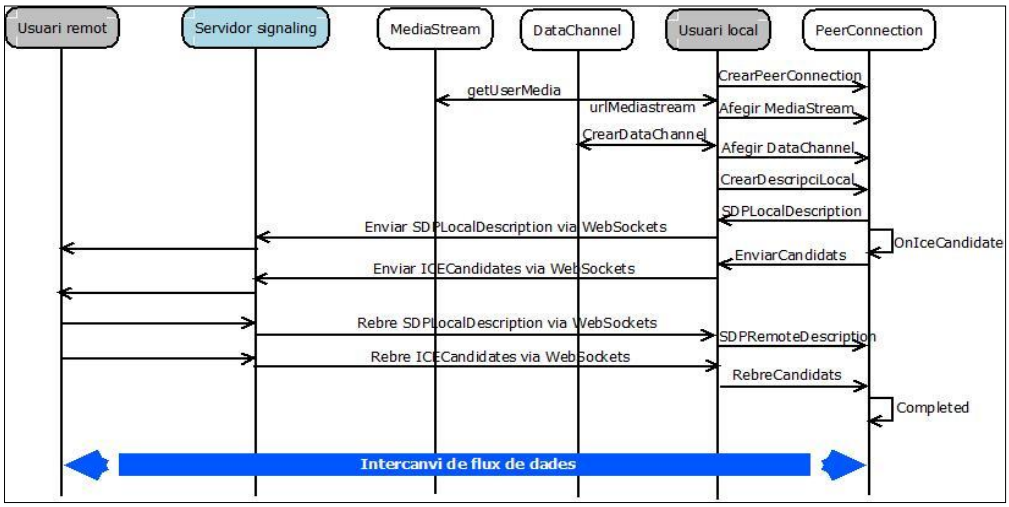


Figura 4

La figura 4 està enfocada sobre la perspectiva de l'usuari local, mostrant el procediment general que ha de seguir per obtenir el flux dels dispositius locals i intercanviar-los amb un usuari remot.

Pel que fa l'usuari remot, el procediment és el mateix, excepte que ell, quan s'intercanviïn els continguts **SDP** (l'SDP es el format en que es descriu els paràmetres d'inicialització de l'objecte MediaStream (restriccions de sessió)) primer rebrà i crearà la descripció remota amb l'SDP que li ha enviat l'usuari local i llavors crearà el seu SDP i l'enviarà. Per ser exactes, el primer crida el mètode *createOffer()*, que li retorna un SDP de tipus "Offer" i l'envia, i l'altre usuari, al rebre l'SDP de tipus offer, respon enviant el seu objecte SDP de tipus "answer", que l'obté cridant el mètode *createAnswer()* amb les restriccions pertinents.

D'aquesta manera, podem descriure el procediment d'obtenció dels dispositius locals i intercanvi d'objectes SDP, fins a obtenir accés a rebre dades dels dispositius remots de la següent manera:

1. *Creació de l'objecte MediaStream a partir del teu dispositiu local (micròfon, càmera ...).*
2. *Obtenció de una URL única que identifica l'objecte MediaStream local que hem creat.*
3. *Utilització de la URL obtinguda per a mostrar el flux del MediaStream localment.*
4. *Creació de l'objecte PeerConnection (canal bidireccional sobre el que s'intercanviaran els MediaStreams). Cal establir un PeerConnection per a cadascun dels usuaris amb els que vulguem parlar.*
5. *Afegir l'objecte MediaStream local al PeerConnection per enviar-lo cap a l'altre usuari.*
6. *Enviar un missatge descrivint les característiques de la teva sessió indicant el nom de la sessió, descripció de temps, velocitats de transmissió... (protocol Signaling)*
7. *Rebre la descripció de la sessió de l'altre peer (usuari). (protocol Signaling)*
8. *Procés de la descripció de l'usuari remot i l'afegiment del vídeo remot al teu PeerConnection.*
9. *Obtenir la URL dels streams remots.*
10. *Usar aquesta URL mostrar el flux del MediaStream remot que rebem.*

Al final del procediment, l'usuari local tindrà un objecte *PeerConnection* amb una URL única per l'objecte MediaStream local (sobre la qual obtindrà i enviarà dades) **i una URL única per l'objecte MediaStream remot** (sobre la qual rebrà el flux que li envia l'altre usuari a través del seu objecte PeerConnection).

Encara que es dibuixi de manera seqüencial en el diagrama de seqüència, paral·lelament al procés d'intercanvi d'objectes SDP es fa el intercanvi de candidats, un cop l'usuari crida el mètode *createOffer()* o *createAnswer()* per obtenir el seu objecte SDP, l'ICE Agent comença de fons la búsqueda de candidats. Cada cop que troba un possible candidat (un port i @IP per on enviar i rebre el flux d'un o varis objectes *MediaStreamTrack*) llança l'event *onIceCandidate* i s'envia el candidat al *PeerConnection* remot mitjançant un canal de WebSockets, al rebre'l, l'usuari remot l'afegeix al SDP mitjançant el mètode ***addIceCandidate*** i l'ICE Agent remot mira si pot establir canal entre un dels seus candidats i el candidat que ha rebut.

Quan troben almenys un canal (dos ICE candidates) pel intercanvi de cada flux multimèdia es passa a un estat de ***completed*** i comença la conversa.

És important tenir en compte que **en una descripció SDP**, com després veurem, **i van incrustats els ICE candidates. Però si enviem l'SDP abans que es comencin a enviar candidats com ho fa?**

De fet tenim dos opcions. Els navegadors actuals usen una extensió del ICE Agent, el **Trickle ICE Agent** [8]. Aquesta extensió permet que l'usuari remot pugui anar comprovant els candidats a mesura els va rebent, pues un **ICE Agent** normal ha de tenir tots els candidats abans no poder començar a comprovar quins candidats són factibles per envia'ls-hi els fluxos.

Actualment, amb els Tickle ICE Candidates, té sentit enviar els candidats de manera separada a l'objecte SDP fent així que, quan l'usuari remot rep les dades, com que ja ha rebut ell el SDP, sigui ell qui afegeixi els candidats al SDP a mesura els va rebent. Però **el procés de senyalització es pot fer sense enviar cap ICE Candidate via WebSocket**, cosa molt útil amb els **ICE Agents** corrents, però no tant útil amb els navegadors actuals.

Per fer-ho només cal, un cop creat l'SDP, esperar a enviar-lo quan l'ICE Agent hagi recuperat tots els candidats. Si fem això, com que WebRTC ja haurà procurat d'afegir els candidats a la descripció de sessió, quan enviem l'SDP aquet ja portarà tots els candidats, i no caldrà fer servir el mètode ***addIceCandidate***.

Com hem anticipat abans, **en cas de que l'usuari tingui @IP o ports que no són públics a Internet l'ICE Agent necessita un servidor STUN per a poder buscar candidats visibles des de l'exterior. I en el cas extrem de no trobar-los li fa falta un servidor TURN per on redirigir la conversa.**

Seguidament es mostrarà una descripció d'aquests dos servidors (STUN i TURN) tant necessaris per WebRTC, però que també es poden usar per altres contextos relacionats en la cerca de canals P2P i re-direccions d'aquests.

DESCRIPCIÓ DELS SERVIDORS STUN I TURN [3]

El servidor STUN (*Session Traversal Utilities for NAT)* i el seu protocol permeten a l'ICE Agent saber a quin tipus de xarxa es troba i , en cas de xarxes privades, descobrir les @IP i ports públics. En aquest aspecte el servidor STUN pot distingir entre tres grans tipus, xarxes públiques, xarxes privades darrere d'un NAT i xarxes restringides per un Firewall.

El servidor TURN (*Traversal Using Relays around NAT)* és un servidor de reenviament de dades que es proporciona al ICE Agent per tal de que, si no es troba cap @IP i port a les dos bandes pel qual els dos usuaris puguin enviar dades d'igual a igual (degut a restriccions del Firewall o NATS que restringeixen l'entrada a l'exterior), puguin fer-ho fent passar un canal pel servidor, que en aquest cas lo únic que farà es reenviar el que rep sense cap processament.

D'aquesta manera, la combinació dels DOS servidors permetrà establir un canal sigui quin sigui l'escenari i complexitat.

FUNCIONAMENT DEL SERVIDOR STUN [5]

Per tal d'entendre millor el funcionament del servidor STUN primer es necessari conèixer que és un router NAT i els tipus de router NAT que hi ha:

NAT (Network Address Translation)

Un router NAT permet crear @IP úniques segons un abast (scope) determinat, mitjançant les traduccions d'@IP i jugant amb els ports que juntament amb les @IP ens permeten identificar un procés. És a dir, els routers NAT permeten amagar xarxes senceres sota una sola @IP pública usant una taula de traducció que tradueix una @IP i port de la xarxa interna a la @IP pública i qualsevol port que assoleixi.

El problema dels NATS són que un usuari de la xarxa pública no podrà començar a parlar amb un usuari que estigui a una xarxa privada darrere d'un NAT si abans l'usuari de la xarxa privada no ha iniciat la conversa.

A partir d'aquest aspecte es poden distingir quatre tipus de NAT que amb l'ajuda del servidor STUN l'ICE Agent es capaç de distingir:

FULL-CONE NAT

Amb un NAT d'aquest tipus, quan s'estableix una entrada a la taula NAT del router, assignant a una interfície i port de la xarxa local la interfície pública del router i el port escollit pel router, a partir d'aquell moment en que es crea l'entrada **qualsevol usuari de la xarxa externa podrà enviar paquets cap a la nostre aplicació que té la @IP i port externs assignats pel router**.

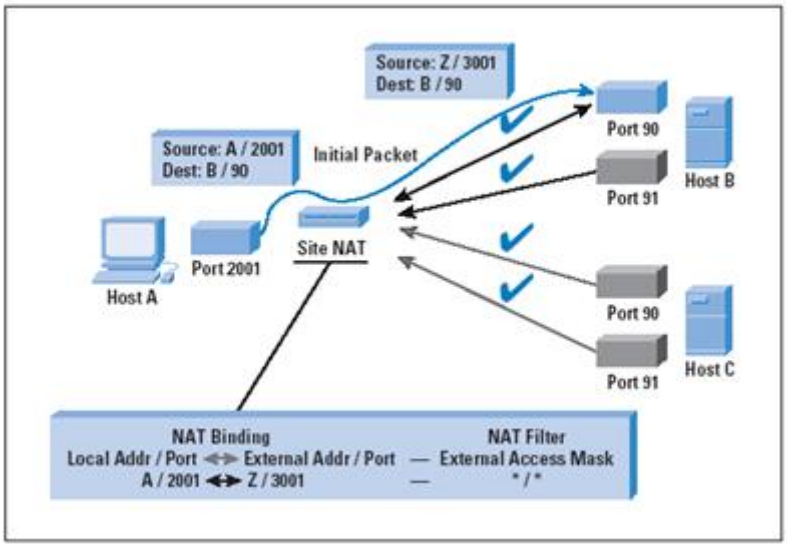


Figura 5

Així doncs, un router amb aquest NAT té la característica que tots els paquets de la mateixa direcció i mateix port interns són mapejats a la taula NAT amb la mateixa direcció i ports externs. Llavors, qualsevol host extern podrà enviar un paquet al host intern enviant-lo a l'adreça i port extern amb el que ha estat mapejat.

(ADDRESS) RESTRICTED-CONE NAT

Al igual que abans, quan una aplicació d'un host intern parla amb la mateixa aplicació d'un host extern es crea una nova entrada a la taula NAT, però aquest NAT és més restrictiu, ara **només podran enviar paquets a l'aplicació del host intern (utilitzant l'entrada creada) els hosts externs que abans el host intern els havia parlat**.

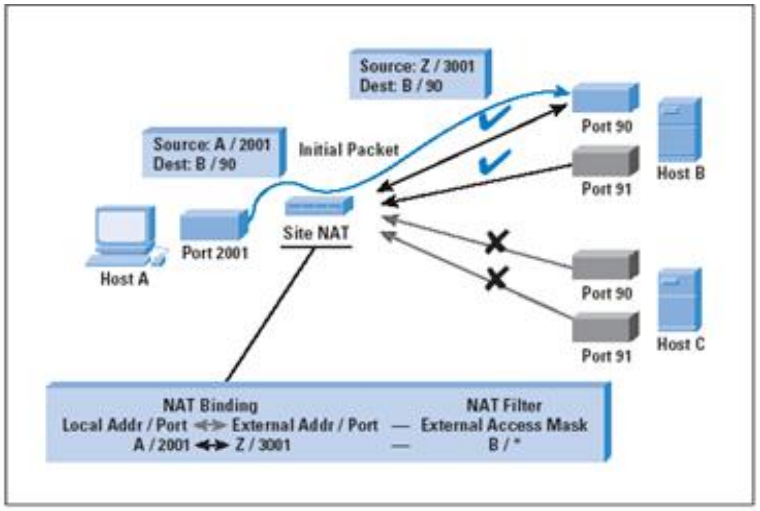


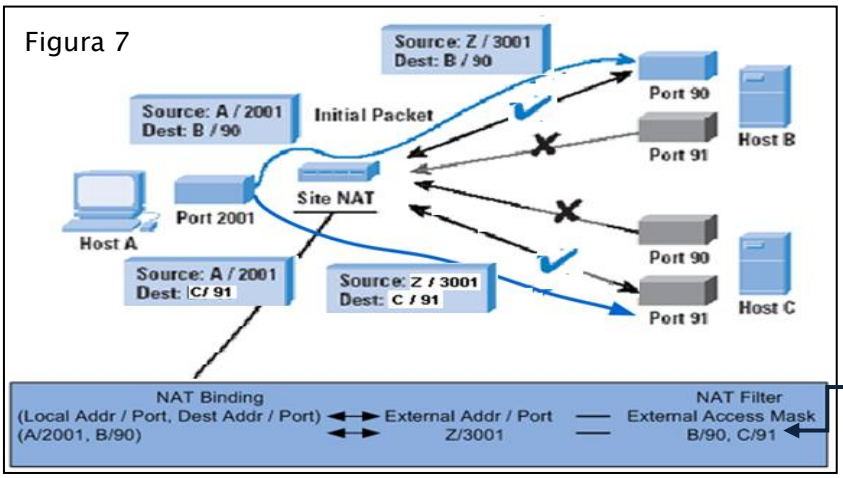
Figura 6

Així doncs, si nosaltres som un host intern i iniciem una conversa amb un host extern es guardarà la @IP del lloc amb el que hem iniciat conversa i llavors, a partir d'aquest moment i fins que s'esborri l'entrada de la taula NAT, el host extern amb aquella @IP podrà enviar paquets al host intern a la @IP i ports públics assignats a l'aplicació del host intern (encara que els ports origen el host extern siguin diferents). Però la resta de hosts amb diferent @IP no podran enviar paquets a través d'aquella entrada.

PORT- RESTRICTED-CONE NAT

Fa exactament igual que el NAT RESTRICTED-CONE, però aquest encara és més restrictiu ja que només permet que entrin paquets d'aquelles aplicacions amb @IP i ports externs que nosaltres abans (amb la mateixa aplicació interna) havíem iniciat l'intercanvi de paquets.

Figura 7



Com podem veure, cada cop que volem parlar des de la mateixa aplicació interna amb un usuari extern amb @IP i/o ports diferents hem de ser nosaltres qui iniciem la conversa, tot i així, només es crearà una entrada a la taula NAT i el que s'afegiran cada cop són permisos de restriccions.

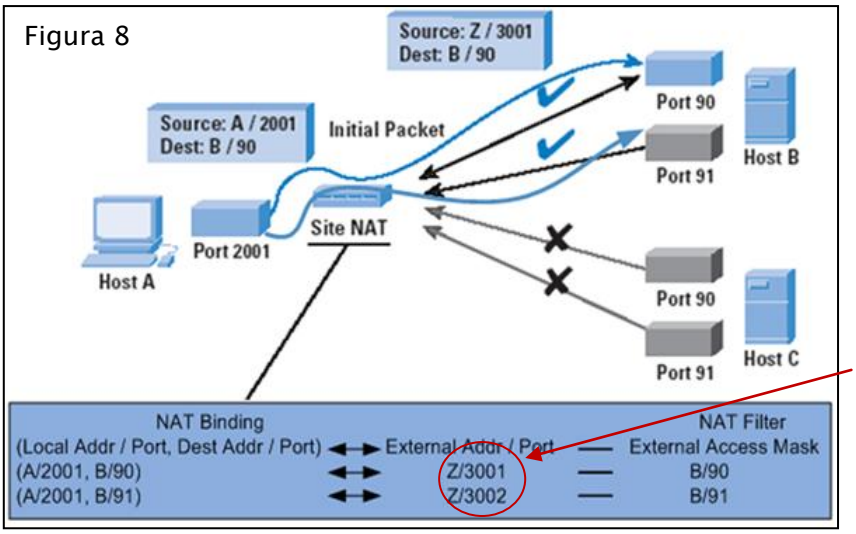
Així doncs en aquest NAT la restricció inclou també números de ports. De manera que un host extern (amb @IP X.X.X.X i port P) només pot enviar un paquet a l'aplicació interna si prèviament el host intern li havia enviat un paquet a l'adreça IP X.X.X.X i port P.

SYMMETRIC NAT

Cada sol·licitud de la mateixa adreça IP i port interns cap a una adreça IP i port destí concrets es correspon amb una única adreça i port origen externs. Si el mateix ordinador intern envia un

paquet, encara que sigui amb la mateixa adreça i port origen, però cap a un destí diferent, s'utilitza una correspondència diferent.

Figura 8



Tot i ser la mateixa aplicació al enviar paquets a @IP i ports diferents cada cop es crea una nova entrada amb un nou port (i/o @IP si el NAT és dinàmic) .

En aquest cas, al parlar amb la mateixa aplicació interna amb els usuaris externs @IP B i pots 90 i 91 es creen dos entrades a la taula NAT, una amb port 3001 i l'altre 3002. Això amb el NAT port restrictiu no passaria.

Si el mateix host intern envia un paquet amb la mateixa direcció interna i port a un destí diferent s'usarà un mapeig diferent. Només el host extern que rep el paquet podrà enviar un paquet UDP de tornada al host intern. Així doncs és com el port-restricted-nat però cada cop crea una correspondència diferent.

Però si mirem un NAT des de fora el podem diferenciar perquè treballi de manera **estàtica**, **dinàmica** o **sobrecarregada**. En un **router NAT estàtic**, una direcció IP privada es tradueix sempre a una mateixa direcció IP pública, és a dir, per cada host que tinguem a la xarxa IP privada haurem de fer una entrada a la taula NAT que per aquella @IP privada tingui una @IP pública única. Com podem veure, tindrem tantes @IP públiques com privades.

El router NAT dinàmic té assignades varies direccions IP públiques, de manera que cada direcció IP privada és mapeja utilitzant una de les direccions IP públiques del router. D'aquesta manera quan el router amb NAT emmascara les @IP i ports privats per @IP i ports públics, l'identificador públic podrà variar tan en l'adreça @IP com en el port públics. Tenint en compte això, haurem de seguir tenint per cada host privat engegat una equivalència amb una @IP pública diferent, però a diferència d'abans si tenim 10 hosts i sabem que com a molt 5 estan connectats simultàniament podem assignar les @IP públiques dinàmicament als hosts que estan en línia i així necessitarem només cinc adreces IP públiques (és una mica l'idea del protocol i servidors DHCP).

El **NAT per sobrecarrega** és el més comú de tots, de fet és el que s'utilitza avui en dia en les xarxes privades de les cases. Amb aquest NAT es poden mapejar múltiples adreces IP privades a través d'una única @IP pública; d'aquesta manera no haurem de contractar més d'una direcció IP publica i així és mes barat i no gastem @IP públiques.

És a dir, el router, quan emmascara les @IP i ports privats per @IP i ports públics, l'identificador públic sempre tindrà la mateixa @IP i lo únic que variarà serà el port, així múltiples hosts de la xarxa privada surten tots a Internet amb una única @IP pública.

Tot i tenir aquesta segona classificació de tres tipus de routers, a fins pràctics, el servidor STUN segueix mirant segons l'altre classificació de quatre, ja que lo únic que li interessa aconseguir és la @IP i port amb els que es veu la nostre aplicació a l'exterior i com s'ho pot fer l'ICE Agent per intentar evitat l'ús del servidor Turn.

SERVIDOR STUN

El servidor STUN (Session Traversal Utilities for NAT), i el seu protocol client/servidor que funciona sobre UDP (en el cas de TCP s'afegeixen alguns missatges més, però el funcionament és molt semblant, en aquest cas ens centrarem amb UDP) [4] , permeten a l'ICE Agent de WebRTC obtenir:

- Direcció IP pública del router.
- Port extern del router associat al port intern del client dins la NAT.
- Tipus de NAT en la que es troba.

El servidor **STUN necessita dos @IP i dos ports diferents (un per cada una de les dos adreces IP) per tal de que pugui descobrir sota quin tipus de NAT es troba el client.**

De fet, no és realment el servidor STUN qui descobreix tot això sinó que és el l'ICE Agent (mitjançant l'intercanvi de missatges del protocol STUN amb el servidor).

El client (ICE Agent) va fent preguntes i segons les respostes que rep o deixa de rebre del servidor STUN dedueix la seva @IP i port públics juntament amb el tipus de NAT que l'encobreix.

Anem a veure quines preguntes es fan i com funciona tot :

PREGUNTA 1: STUN TINC UN NAT?

La primera pregunta important a fer és si realment estem sota un NAT o la nostra interfície és pública. Per saber-ho el que fem és demanar a la @IP i port primaris del servidor STUN (amb un missatge ***Binding Request***) que ens retorni un missatge indicant quina és la @IP i el port origen que tenia el missatge que hem enviat.

El servidor STUN ens contestarà amb un missatge ***Binding Success Respose*** on en el camp MAPPED-ADDRESS ens indicarà amb quina @IP i port ell a rebut el paquet de petició.

Així doncs el host es mirarà el camp MAPPED-ADDRESS i ens podem trobar en dos casos:

1- Que la adreça del camp MAPPED-ADDRESS coincideixi amb la del client:
En aquest cas no tenim NAT i cal que mirem si tenim un firewall simètric que pugui dificultar l'accés.

2- Que la adreça no coincideixi amb la del client:
En aquest cas tenim un router que fa NAT i ara que ja sabem la @IP i port públic amb el que ens veuen cal saber sota quin tipus de router NAT ESTEM (full-cone, restricted-cone, symetric NAT o port-restricted-cone)

PREGUNTA 2 (CAS DE QUE NO TENIM NAT): STUN TINC UN FIREWALL?

Per saber si tenim o no un Firewall el que fem és demanar al servidor STUN que ens retorni un missatge amb diferent @IP i port a la qual fem la petició. Així doncs el servidor ens respondrà amb la un missatge ***Binding Response*** a través de la @IP i port secundaris que té.

Si arriba no hi ha Firewall simètric, si no arriba **tenim un Firewall**.

PREGUNTA 2 (CAS DE QUE TENIM NAT): TENIM UN NAT, QUINES PREGUNTES FEM ARA...

PREGUNTA 3: TENIM UN NAT FULL-CONE?

Per saber-ho fem una petició a la @IP i port principal de l'STUN i demanem que ens respongui per l'@IP i port secundaris de l'STUN amb el camp **CHANGE-REQUEST**.

1- Si arriba alguna cosa tenim un **NAT FULL CONE!** Ja que qualsevol interfície amb una adreça IP i ports públics no registrats a la taula NAT poden enviar paquets a traves de la connexió que hem obert.

2- Si no arriba tenim un NAT restrictiu o simètric. Ja que no ens arriben els paquets a l'aplicació si no són estrictament del host amb @IP i port amb el qual nosaltres abans hem començat a parlar.

PREGUNTA 4: TENIM UN NAT SIMÈTRIC?

Per saber-ho cal que fem una petició des del client a la @IP i port secundaris del STUN, d'aquesta manera, si quan ens retorna el missatge amb el camp MAPPED-ADDRESS emplenat veiem que tot i utilitzar la mateixa aplicació interna el port ha canviat respecte el port que teníem quan hem ens ha respost la primera petició en la pregunta 1, vol dir que cada cop (encara que l'aplicació ja tingues una traducció a la taula per un altre destí) el router NAT fa un mapeig diferent (crea una nova entrada amb un nou port).

Així doncs, quan rebem la resposta mirem el camp mapped address i podem tenir dos situacions:

1- Si la @IP i port no coincideixen amb els que hem rebut en la primera pregunta de totes estem darrera d'un **NAT SIMÈTIC!**

2- Si coincideix encara no estem. Ens queda una pregunta per saber si el NAT es restrictiu o port restrictiu.

PREGUNTA 5: QUIN DELS 2 NAT's RESTRICTUS TENIM?

Demanem (un ***Binding Request***) a la @IP primària i port primari del servidor STUN que ens enviï un missatge des de la @IP primària i port secundari mitjançant el camp **CHANGE-REQUEST** de nou.

1- Si arriba un ***Binding Response*** estem en un **NAT RESTRICTIU!** Ja que si rebem un paquet per el canal obert del mateix host, però amb un port diferent amb el qual el host privat havia iniciat la conversa el router NAT l'accepta igual.

2- Si no arriba serà un **NAT RESTRICTU AMB PORT!** Ja que el router NAT no accepta per l'entrada creada cap altre paquet que no vingui de l'adreça IP i port amb els quals havia iniciat conversa.

Si ens mirem tot l'intercanvi de missatges explicats anteriorment de manera gràfica tenim:

Figura 9

L'ICE Agent, realitza aquest intercanvi de missatges amb el servidor STUN per cada un dels objectes *MediaStreamTrack*, és a dir, per cada flux de dades que es voldrà enviar, un cop ha finalitzat el protocol per a obtenir la @IP i ports públics d'un MediaStreamTrack, juntament amb el tipus de NAT, llança l'event ***addIceCandidate*** per notificar a l'objecte PeerConnection del nou candidat.

Però els candidats que ens proporciona el protocol STUN podrien no ser suficients en alguns casos. De fet, els ICE Candidats que s'obtenen a partir d'aquest protocol només funcionen si cap, o només un dels 2 usuaris estan en una xarxa privada; O si els 2 usuaris estan en xarxes privades però almenys un dels 2 usuaris té les entrades SRTP (Secure Real-Time Transport Protocol) obertes per l'altre.

Aquest segon cas passarà sempre si un dels usuaris té un router *Full-cone*. I es pot donar el cas de que encara no s'hagi esborrat l'entrada a la taula NAT en els routers *Restricted-cone*, cosa que també permetria poder establir una connexió directe entre els dos usuaris.

Tenint en compte el funcionament dels routers NAT, **pot ser que ens trobem en el problema de que, malgrat enviar a l'usuari remot un *ICE Candidate* amb @IP i ports públics, les dos bandes de la connexió WebRTC siguin incapaços d'establir un canal per tots els *traks***. Això es degut al

fet de que cap dels dos usuaris és capaç de començar a enviar paquets degut a l'impossibilitat de poder sobrepassar el NAT remot sense que ell abans li sobrepassi el seu NAT.

En aquests casos es quan apareix la necessitat d'usar ICE candidates obtinguts gràcies al servidor TURN.

FUNCIONAMENT DEL SERVIDOR TURN

Quan es vol establir una connexió WebRTC, la principal prioritat és que aquesta sigui d'estil P2P, però alguns cops, degut a NATs o Firewalls, és impossible. L'ICE Agent, quan envia el candidats a través del protocol SDP o a través d'un canal extern (com en el nostre cas) els ordena per ordre de complexitat, posant primer els candidat de tipus local (tant UDP com TCP), posteriorment els candidats de tipus STUN (aquests candidats, els locals i els STUN permeten connexions P2P) i finalment, envia els candidats de tipus TURN.

Els candidats de tipus TURN fan passar l'intercanvi de fluxos entre els 2 usuaris per el servidor de reenviament TURN, convertint WebRTC a una aplicació client/servidor.

Per tant passem d'aquest escenari en que podríem establir la connexió peer to peer normal entre els dos clients un cop havíem consultat la @IP i ports públics al servidor STUN i havíem fet l'intercanvi de candidats.

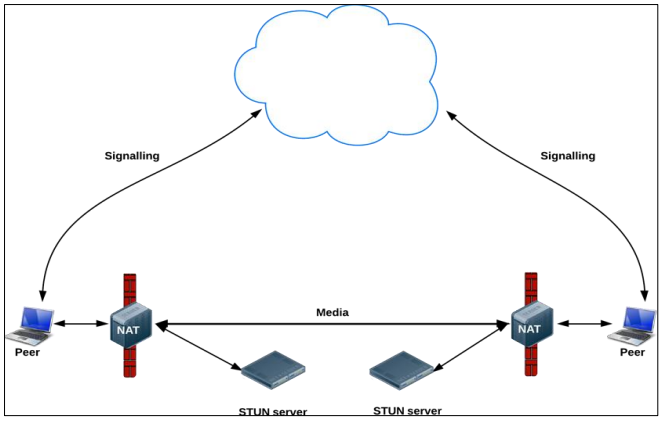


Figura 10

A aquest nou escenari en que la connexió peer to peer no és possible i és necessari la utilització del servidor TURN, per el qual han de passar tots els missatges entre els dos hosts, i que l'únic que fa es reenviar els paques que rep d'un host cap a l'altre:

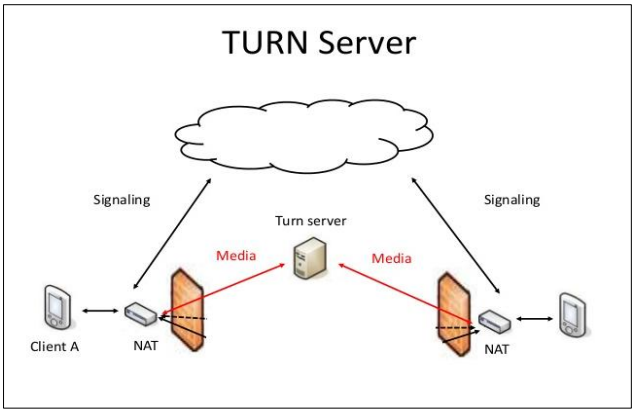


Figura 11

No cal dir que **l'eficiència en l'intercanvi de fluxos d'àudio, vídeo i dades en el segon escenari és molt més lent i ineficient que el primer.**

El funcionament del servidor "TURN" és el següent: [6]

Quan els 2 usuaris no poden usar els candidats P2P usen el candidat del servidor TURN. Per fer-ho és necessari que un dels dos usuaris obri un port al servidor TURN per tal de poder enviar les dades per allà quan l'altre usuari s'inscrigui en aquest port.

Així doncs **el primer que s'ha de fer és obrir un port al servidor TURN i per fer-ho se segueix la següent seqüència de missatges:**

1- UN DELS DOS CLIENTS DEMANA L'OBERTURA D'UN PORT AL SERVIDOR TURN:

Els missatges que s'intercanvien amb el servidor TURN per l'obertura del port són els següents:

Petició

- **Allocate request** - El client demana al servidor que obri un port

Resposta

Normalment els servidors TURN tenen un usuari i password que s'han d'indicar correctament alhora de fer la petició i per això la resposta pot ser:

- **Allocate response** - El servidor respon al client indican quin és el port obert
- **Allocate error** - El servidor no dona permís al client per obrir el port (pot ser perquè no hem posat correctament l'usuari i contrasenya).

Un exemple del diagrama de seqüència alhora d'iniciar la cessió podria ser el de la *figura 12*:

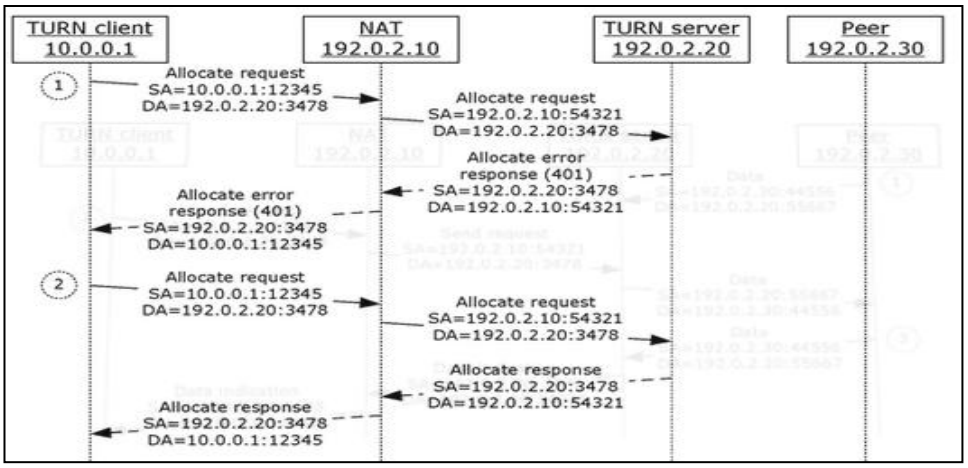


Figura 12

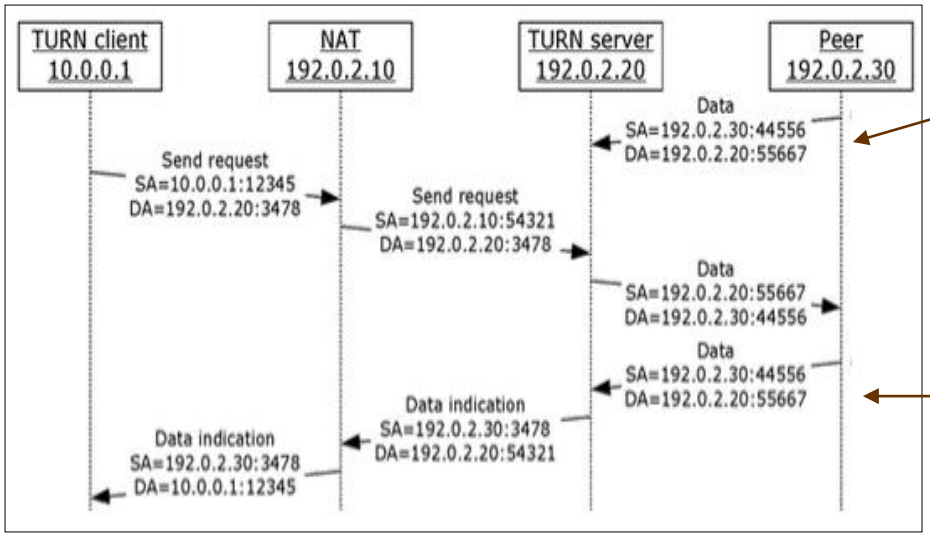
2- ARA JA ES PODEN COMENÇAR A INTERCANVIAR MISSATGES:

El host que ha iniciat sessió al TURN enviarà paquets de ***Send request*** (on en el camp send adress indiquem la @IP i port públics de l'usuari amb qui volem parlar) per enviar dades i rebrà paquets ***Data Indication*** del servidor amb dades enviades per part de l'altre host.

Per altre banda, l'altre host que no ha iniciat la sessió al TURN enviarà i rebrà paquets del port que se li ha obert al servidor TURN quan l'altre usuari ha fet el primer SEND INDICATION utilitzant el protocol de la capa d'aplicació DTLS.

És important tenir en compte que un cop l'usuari que inicia la sessió ha demanat l'obertura del seu port, l'altre usuari no podrà enviar dades fins que el primer usuari no li enviï un SEND INDICATION i se li obri al servidor TURN un port per ell.

Un exemple de l'intercanvi de missatges (un cop el client ha demanat obrir un canal *WebRTC* i se li ha confirmat l'obertura del port al TURN server) amb un altre usuari podria ser:



Com podem veure, com que el TURN client encara no a enviat el primer paquet amb un SEND INDICATION l'altre usuari encara no pot parlar a través del TURN, tot i així obre l'entrada al NAT..

Ara, l'altre usuari de WEBRTC ja pot enviar missatges quan vulgui al seu company de sala a traves del TURN ja que ja se li ha obert el seu port en el TURN.

Figura 13

Així doncs ja hem vist quina és la solució que tenim perquè WEBRTC funcioni en qualsevol escenari. La gràcia és que *WebRTC* usa els ICE candidates i ens permet declarar un servidor STUN+TURN per tal de que l'intercanvi de candidats i el posterior intercanvi de fluxos de veu, vídeo i dades es pugui fer sense cap problema.

Per declarar el servidor TURN i STUN en el codi és tan senzill com afegir les següents línies a la nostre variable que indica els servidors que s'usen alhora de buscar els candidats i alhora de fer l'intercanvi de fluxos:

```
//configuracio del iceServers per a utilitzar Stun i Turn
var STUN = {url:'stun: @IP:PORT'};
var TURN = {url:'turn:@IP:PORT',credential:'pass',username:'usuari'};

var pc_config = webrtcDetectedBrowser === 'firefox' ?
  {'iceServers': [STUN,TURN]}:
  {'iceServers': [STUN,TURN]};
```

Més informació sobre TURN [7].

PILA DE PROTOCOLS A WebRTC i auxiliars [13]

Els protocols que usa WebRTC, juntament amb es protocols que s'usen per intercanviar dades en el procés de senyalització, es poden esquematitzar de la següent manera segons les diferents capes:

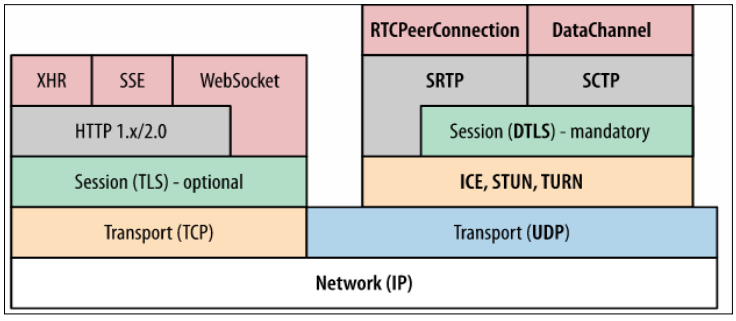


Figura 14

En el dibuix es mostra la pila de capes a partir de la capa de *InterXarxa* a la qual sempre s'usa el protocol d'Internet (IP); Per sota d'aquesta capa, són indiferents els protocols que s'usin, tot i que en WebRTC, per obtenir les millors prestacions de processament i enviament de fluxos, sempre es millor usar el protocol d'Ethernet com a protocol per a la capa física (pel fet de que Ethernet permet injectar més quantitat d'informació per segon a la xarxa).

A partir de la capa de *InterXarxa* en amunt els protocols que es poden usar són variats depenent de les circumstàncies, seguidament s'analitzaran els possibles protocols per a cada capa.

Capa de transport

En aquesta capa, WebRTC (els fluxos multimèdia o dades enviats a través de una connexió PeerConnection) pot usar tant TCP com UDP, tot i que l'ús de ICE candidats de tipus TCP sol ser escollida com a última opció, i només és possible en el cas de que els candidats ICE seguin de tipus locals.

La prioritat de UDP en el intercanvi de flux és degut a que, al ser WebRTC una aplicació en temps real és molt sensible al retard i no tant en la pèrdua de paquets, és a dir, UDP no és ni orientat a la connexió, ni fiable, ni ordenat (coses que proporciona TCP) però a canvi es més ràpid que TCP.

En el lloc on si que s'usa TCP és en el intercanvi de dades amb el servidor (tant per peticions HTTP com intercanvi de dades amb WebSocket). Exceptuant els servidors STUN i TURN, ja que el protocol STUN funciona sobre UDP (User Datagram Protocol).

Entre la capa de transport i aplicació

En aquest punt s'usen els protocols de TLS o DTLS. El protocol TLS s'usa sobre TCP, en especial en WebSockets, proporcionant seguretat contra espies, falsificació i manipulació de missatges a la capa de transport.

Per altre banda DTLS és una extensió de TLS per ser usat sobre UDP, DTLS proporciona totes les característiques de seguretat de TLS, però amés soluciona els problemes de paquets perduts i no reordenació de UDP, afegint a la capa UDP la característica de retransmissió de paquets i assignant una seqüència de números als paquets.

En WebRTC s'usa DTLS sobre UDP alhora d'enviar missatges a través del canal de dades (dataChannel) i d'establir contacte entre els 2 navegadors. És a dir, s'usa DTLS en el intercanvi de missatges entre el servidor STUN i TURN per tal de que el intercanvi de dades sigui segur.

Altrament, com a protocol d'enviament de fluxos multimèdia s'usen els protocols de la capa d'aplicació SRTP, RTCP directament. En aquest cas, DTLS s'usa en els primer missatges de negociació, pet tal de fer un intercanvi segur de la clau que s'usarà per encriptar els missatges en els protocols SRTP i RTCP. Així doncs DTLS s'usa en l'intercanvi de fluxos multimèdia, només per derivar la clau comuna (usant logaritmes modulars) que usaran els 2 navegadors per anar encriptant i desencriptant les dades.

Per sobre de DTLS

Per sobre a DTLS, en el cas de dades enviades amb DataChannel podem trobar el protocol SCTP, que s'usa per encriptar i enviar dades a través del DataChannel, en aquest cas, SCTP treballa sobre de DTLS, que alhora treballa sobre UDP.

Capa d'aplicació

Per la capa d'aplicació podem trobar tres protocols, SRTP, RTCP i Data Channel, els 2 primers van directament a sobre d'UDP i s'usen per l'intercanvi de fluxos d'àudio i vídeo (pels objectes MediaStreamTrack). El tercer, s'usa per enviar dades a través del DataChannel.

El protocol SRTP s'usa per enviar els fluxos multimèdia i RTCP per codificar la informació necessària que necessita saber l'usuari remot sobre aquests fluxos.

Per a cada MediasTreamTrack se sol usar un port diferent. És a dir, si jo envio senyal d'àudio, s'enviarà el flux usant SRTP i les dades del control de flux es codificaran usant RTCP. Però alhora de fer viatjar el flux per la xarxa, els missatges RTCP s'envien dins dels SRTP, permetent així la necessitat d'usar només un port per cada MediastreamTrack.

Tot i la idea d'usar un port SRTP per cada flux multimèdia, en el cas de que els candidats ICE siguin de tipus STUN o TURN, el que se sol fer és unificar tots els paquets SRTP en un únic paquet, enviant totes les dades dins d'un únic objecte SRTP, això permet no haver d'obrir múltiples ports en el servidor TURN o a la taula del router NAT.

5.1.2. Node.JS

NodeJS és un intèrpret *JavaSCript* del costat servidor que canvia la noció de com hauria de treballar un servidor. La seva finalitat és permetre a un programador construir aplicacions altament escalables i escriure codi que gestioni milers de connexions simultànies en una sola màquina física.

Així doncs *NodeJS* et permet programar un servidor usant el llenguatge JavaScript, cosa molt útil per usar la mateixa llibreria de WebSockets que usa el navegador. Això és degut a que Node JS usa el motor V8, el mateix que usa Chrome per executar JavaScript i que permet executar codi a grans velocitats.

El fet de que Node sigui tant escalable es gràcies a que és un entorn d'execució dirigit a esdeveniments (per això es programa amb JavaScript) i per tant és **asíncron**, això significa que implementar APIs REST (cosa que es necessita pel treball) és molt eficient pel fet de que el processament de les peticions s'executa concurrentment.

Una altre gran utilitat de NodeJS és que es poden expandir les funcionalitats inicials de NodeJs important mòduls. Per afegir un mòdul només cal escriure a la consola dels servidor ***sudo npm install "***nomModul***"*** i importar el mòdul en el codi amb el mètode ***require("***nomModul***")***, i a partir d'aquell moment es podran usar el conjunt de mètodes que el mòdul proporciona.

Aquest fet fa que el servidor pugui proporcionar funcionalitats molt variades, per exemple, com es fa en el treball, permet comunicacions Socket.IO i peticions HTTP per a la API REST.

5.1.3. Websocket

WebSocket és una tecnologia que proporciona un canal de comunicació full-duplex sobre un únic Socket TCP, és a dir, estableix una única connexió TCP entre 2 bandes, que permet a les bandes actuar com a emissor i receptor en el mateix canal (podem enviar i rebre per un únic canal).

És una tecnologia per ser usada en aplicacions client/servidor i actualment està sent normalitzada pel W3C. Gràcies a WebSockets es trenca l'esquema d'HTTP i les comunicacions unidireccionals donant la possibilitat de que tant client com servidor puguin parlar d'igual a igual.

La pila de protocols en WebSocket és la que es mostra a la *figura 14*. És a dir, aprofita que les connexions TCP són orientades a la connexió per establir un canal persistent segur (gràcies al protocol TLS) de comunicació.

A partir de WebSocket, va aparèixer ***Socket.IO***, que és la llibreria que s'usa en el treball per a la gestió de perfils d'usuari i pel procés de signaling de WebRTC . ***Socket.IO*** **és una llibreria ,originàriament en JavaScript, per a Node.js, que permet una comunicació bidireccional en temps real entre client i servidor**; per fer això es basa principalment en WebSocket, però també pot usar altres alternatives com *sockets d'Adoble Flash* o *JSNOP polling o long polling en Ajax*, seleccionant la millor alternativa pel client just en temps d'execució.

És important ressaltar que Socket.IO, malgrat derivar de WebsSocket, no suporta interaccions amb altres clients que usin WebSocket estàndard, això es degut a que Socket.IO no és una aplicació de WebSocket, sinó una llibreria de comunicació en temps real que utilitza varius protocols. Tot i així hi han llibreries per usar Socket.IO en altres plataformes diferent a la Web, oferint llibreries, com per exemple en el cas d'Android, que permeten implementar un client Socket.IO en Java [9]. O fins i tot hi han llibreries per a l'ús de Socket.IO en Java a la part servidora [10].

Per al protocol de senyalització de WebRTC, el procés previ a establir una connexió WebRTC entre dos usuaris, és necessari un canal bidireccional entre client i el servidor de senyalització. D'aquesta manera l'ús de *Socket.IO* i *Node.js* permet solucionar aquesta restricció de WebRTC per a poder fer el intercanvi de descripció de sessió entre els dos usuaris de un PeerConnection.

5.2. Àmbit d'utilització de les tecnologies

Fins ara s'ha fet un breu resum de les tecnologies que s'han usat; WebRTC, Node.JS i Socket.IO, però no s'ha concretat en com s'han usat i perquè. En aquest apartat es relacionarà les tecnologies usades amb l'aplicació. Excepte la llibreria de Socket.IO, que va molt lligada a la tecnologia de WebSockets, la resta de llibreries usades i decisions preses s'explicaran en l'apartat set, després d'haver analitzat les funcionalitats del sistema.

La tecnologia WebRTC només està present alhora de realitzar les videoconferències entre usuaris, mentre que l'ús de Socket.IO i la comunicació amb el servidor Node.JS és necessària en totes les planes web de l'aplicació.

L'aplicació consta de tres planes, una de registre, una per a la gestió del perfil de l'usuari, converses i altres funcionalitats, i una plana per a establir conferències múltiples, així doncs WebRTC s'usa en l'última plana, mentre que l'ús de Socket.IO és necessari a totes, tot i que en la última plana, només és necessària pel protocol de senyalització i renegociació d'amples de velocitats de transmissió.

ÀMBIT D'UTILITZACIÓ DE WEBRTC

Partint dels conceptes explicats a l'apartat 5.1.1 s'han usat les llibreries que l'API de WebRTC ofereix per realitzar el sistema de videoconferència. A part, per poder comunicar usuaris que es troben en xarxes privades, s'ha hagut d'oferir a WebRTC un servidor STUN i TURN per a solucionar els problemes que donen els NATs i Firewalls.

Amés d'això, s'ha estudiat el protocol SDP pet tal de poder regular els amples de banda i solucionar els problemes d'escalabilitat a la part client. **Seguidament es mostra com el protocol SDP i com és fa per establir els límits d'ample de banda en els canals multimèdia:**

PROTOCOL SDP

El protocol SDP és el protocol que usa WebRTC per a fer l'intercanvi de sessions de descripció (a les sessions de descripció WebRTC també se'ls sol anomenat durant el document objectes SDP o SDP per tal d'abreviar la sintaxi, malgrat ser SDP el protocol i no la descripció en si).

Quan un usuari crea una *PeerConnection* és necessari que a través del mètode *createOffer* o *createAnswer* defineixi una descripció de sessió on indiqui, pensant en l'altre usuari, les característiques que l'altre usuari necessita saber. En aquestes característiques i ha d'haver tant dades generals, com una descripció per a cada medi multimèdia diferent que es pot usar, que poden ser entre d'altres, el **tipus de *còdecs*** que s'usaran per a la compressió i descompressió dels fluxos multimèdia, les **velocitats de transmissió (VDT)** que poden consumir o les claus de xifrat que han de derivar per obtenir la clau d'encriptació.

Cada un dels dos *peers* (usuaris que han creat l'objecte PeerConnection) ha de crear un objecte SDP i enviar-lo (a través del canal de *Signaling Socket.IO*) cap a el peer remot (el protocol de Signaling bàsic és tant senzill com el reenviament d'aquest missatge).

Aquest objecte SDP està compost de la següent manera (els atributs opcionals estan marcats amb un *):

```
v=  (Versió del protocol: sol haver-hi un 0)
o=  (Origen i identificador de sessió)
s=  (Nom de sessió)
i=* (Informació de la sessió: Títol o breu informació)
u=* (URI de descripció)
e=* (Zero o més @ de correu i opcionalment els noms dels contactes)
p=* (Números de telèfon i opcionalment els noms dels contactes)
c=* (Informació de connexió, se sol incloure en els medis)
b=* (Zero o més línies amb informació d'ample de banda, se sol incloure
       només en la descripció dels medis)
Una o més línies de descripció de temps
z=* (Ajustos de zona horària)
k=* (Clau de xifrat, si no es posa aquí s'ha de posar a la descripció
       dels medis)
a=* (Zero o més línies d'atributs de sessió)
Zero o més descripcions de medis (poden ser medis d'àudio, vídeo, o de
compartició de pantalla)
```

```
t=  (temps durant el qual la sessió estarà activa)
r=* (número indicant els cops de repetició de t)
```

```
m=  (Nom del medi i port de la capa de transport)
i=* (Títol del medi o breu informació)
c=* (Informació de la connexió (si no s'ha declarat a nivell de
       sessió)
b=* (Zero o més línies d'informació d'ample de banda)
k=* (Clau de xifrat)
a=* (Zero o més línies d'atributs de sessió)
```

La informació ha estat extreta dels següents enllaços [11], [12].

Si mirem l'enllaç [12] és mostra un exemple complet d'un objecte SDP explicant detalladament cadascun dels atributs que s'acaben de descriure.

En el projecte només s'ha treballat amb la part del SDP que descriu cada un dels medis (àudio, vídeo o compartició de pantalla), així que la resta de paràmetres, que molts d'ells també es poden modificar no s'han tocat.

Tal com es mostra en el protocol, per cada medi que volem intercanviar a través del *PeerConnection* usant objectes *MediaStreamTrack*, és necessari afegir aquesta part a l'objecte SDP . Això permet a l'altre usuari conèixer les característiques de com nosaltres tractem aquest medi, juntament amb els ports pels quals el podem rebre.

```
m=  (Nom del medi i port de la capa de transport)
i=* (Títol del medi o breu informació)
c=* (Informació de la connexió (si no s'ha declarat a nivell de sessió)
b=* (Zero o més línies d'informació d'ample de banda)
k=* (Clau de xifrat)
a=* (Zero o més línies d'atributs de sessió)
```

Com podem veure, de tots els atributs que descriuen un *medi*, només el primer és obligatori, aquest camp (***m***) conté el port UDP pel qual es pot rebre el flux d'un track i el nom del tipus de flux multimèdia que s'envia i rep pel *MediaStreamTrack*, que pot ser {*àudio*, *vídeo* o *screen*}.

El segon (***i***) és poc important.

El tercer atribut, **c**, conté la @IP per on l'usuari creador del SDP espera enviar i rebre el tràfic, en WebRTC però al tenir els ICE candidates, aquests tenen preferència sobre aquesta @IP i sol haver-hi un 0 a l'@ IP.

El quart atribut (b) si que és molt important. **En l'atribut b de cada *medi* és on s'indica la velocitat de transmissió màxima a la que l'usuari remot ha d'enviar (i nosaltres rebre) dades del medi marcat en el paràmetre *m*.** És a dir, si per exemple tenim:

```
m=video 60372 UDP/TLS/RTP/SAVPF 100 101 116 117 96
b=AS:550
```

Per indicar un número màxim d'ample de banda es modifica l'atribut b posant b=AS:"numero al que es limita"

En aquest cas estem indicant que l'usuari local no pot rebre les dades dels flux de vídeo a més de 550kbits/s.

El cinquè atribut (**k**) indica la clau local que l'usuari remot a de derivar perquè, un cop aquest usuari hagi derivat la clau de l'altre, els dos tinguin la mateixa clau i puguin aplicar una funció involutiva per l'encriptació i desencriptació de flux.

L'últim atribut (**a**) conté tots els atributs que descriuen la sessió del medi, que bàsicament són còdecs usats, paràmetres ICE, ICE Candidates pels quals es poden rebre les dades del medi, i paràmetres DTLS. Pel que fa els ICE candidates de l'usuari local; En el cas de que s'enviï l'SDP sense haver completat el procés d'obtenció de tots els candidats serà necessari enviar els candidats que falten a través del servidor de signaling, que és el que es fa en el nostre sistema.

Respecte els còdecs utilitzats, tenen més relació amb les VDT del que sembla, els **còdecs** permeten comprimir la informació dels fluxos multimèdia que s'envien a través de la xarxa. En aquest aspecte, si es vol reduir el màxim la VDT necessària cal utilitzar el còdec que més comprimeixi el flux inicial. Però en l'intent de comprimir al màxim la VDT cap tenir en compte també el consum de recursos que usa (en especial per a dispositius no connectats al corrent) i el suport que el còdec dona, ja que no tot els còdecs estan implementats en tots els SO i navegadors.

En el nostre cas els còdecs que s'utilitzen en àudio i vídeo són diferents i són els que *WebRTC* proporciona el màxim suport. El còdec usat per la codificació i descodificació de fluxos d'imatges és el VP8 i per l'àudio es fa servir ISAC. En el treball no s'ha entrat a valorar quins còdecs podrien ser millors que d'altres però altres còdecs com H.264 pel vídeo poder podrien proporcionar millors funcionalitats que VP8.

Tornant amb el 4rt atribut del SDP, per descriure la velocitat de transmissió màxima a la que podem rebre les dades d'un *MediaStreamTrack* d'aquell medi (un *MediaStreamTrack* és l'objecte WebRTC que permet enviar i rebre dades d'un medi determinat), s'afegeix aquest atribut. Aquest atribut farà que l'usuari remot no enviï una quantitat de dades major al numero indicant, limitant així el seu ample de banda de pujada i el nostre de baixada.

És a dir, quan un dels dos usuaris utilitza un *MediaStreamTrack* per enviar i rebre dades d'un medi determinat, l'altre usuari encara que no tingui cap track d'aquell tipus de medi, està obligat a descriure al SDP les característiques del medi (per demostrar això s'han agafat diferents tipus de *PeerConnections* i s'ha vist que per descripcions de sessió sense enviament d'àudio o vídeo es definien els medis d'àudio i vídeo igualment). Tenint així com a mínim una descripció diferent per cada medi.

Desconec el funcionament en casos de que es comparteixin múltiples fluxos d'un mateix medi en una mateixa PeerConnection, però si es comparteix un únic senyal de vídeo o àudio en l'SDP es defineixen els atributs que hem dit per cada medi diferent.

Així doncs donat un medi (per exemple de vídeo) podem trobar-nos en tres casos:

-*Els dos usuaris tenen un track per enviar d'aquell medi (tots 2 envien senyal de vídeo).*

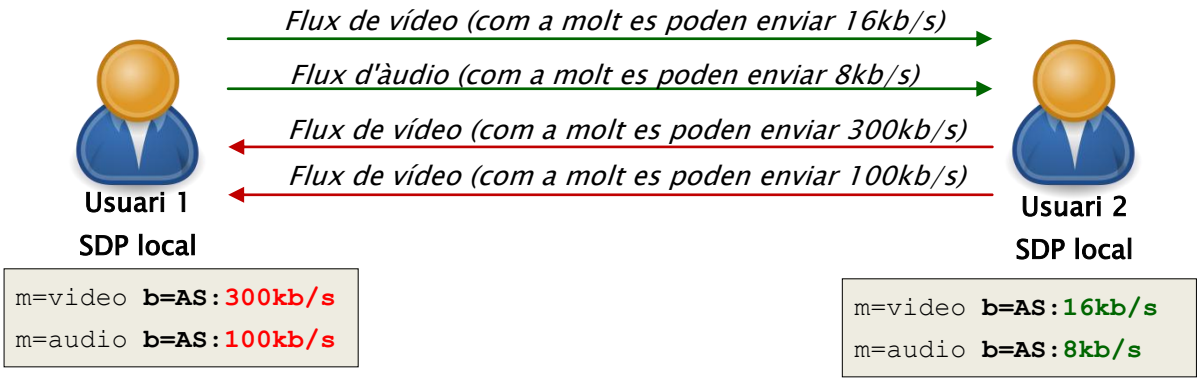
-*Només un usuari envia el flux del medi, l'altre només el rep.*

-*Cap dels dos usuaris envia contingut d'aquell medi de comunicació.*

En tots els casos, en l'SDP local es definiran, almenys la línia ***m***, pel medi d'àudio i vídeo amb certs atributs, encara que no anem a enviar cap tipus de senyal d'aquell medi; llavors, si realment es té un objecte *MediaStreamTrack,* en l'objecte *MediaStream* local, per enviar dades del flux, s'afegiran més paràmetres en el camp ***a*** del SDP descrivint-lo. Per exemple, si enviem vídeo hi ha un camp que posa ***a=sendrecv***, altrament, si no ho fem, només posa ***a=rcvonly***, aquest camp indica quina presencia tenim nosaltres al medi, si rebem i enviem o si únicament podem rebre.

A partir d'aquí, **per afegir la restricció de velocitat de transmissió d'un medi s'afegeix l'atribut *b=AS:"numero de limitació"* per a cada medi diferent**, indicant la velocitats de transmissió de pujada pels *MediaStreamTrack* de l'usuari remot i per tant la baixada del seu flux cap el nostre ordinador.

Seguidament es mostra un dibuix explicatiu:



En aquest cas l'usuari 1 crea el seu objecte SDP dient que ell només vol rebre 300kb/s de fluxos de dades i control de vídeo i 100kb/s d'àudio. Per altre banda l'usuari 2 té unes restriccions més elevades, dient que ell només processarà 16kb/s de dades provinents del medi d'àudio i 8kb/s pel vídeo.

Un cop els 2 usuaris han definit l'SDP se l'intercanvien (amb el protocol de *signaling*) per tal de que l'altre conegui les restriccions.

Després del intercanvi l'usuari 1 veu que l'usuari 2 només pot processar un flux de dades total de 24 kbits/s, això significa que ell no pot enviar més de 24kbits/s i per tant que la velocitat de transmissió de pujada que consumirà serà de 24kbits/s, a conseqüència la VDT de l'usuari 2 serà de 24kbits/s.

Per altre banda, l'usuari 2 se li ordena enviar un flux multimèdia per sota de 400kb/s, volent dir això que la velocitat de transmissió de pujada que consumirà l'usuari 2 serà de 400kb/s i la de baixada del primer usuari serà de 400kb/s.

SRTM (acrònim de la plana Web) limita els amples de banda tenint en compte això. Cada usuari té definit un ample de banda de pujada i baixada per a cada medi. Per exemple pot tenir 32kb/s de VDT de pujada d'àudio, 300kb/s de pujada de vídeo i per la baixada té permesos 60kb/s i 800kb/s pels medis d'àudio i vídeo respectivament.

Degut a que les connexions dels usuaris corrents solen ser asimètriques, i que la velocitat de transmissió de pujada sempre es bastant menor a la de baixada s'ha de ser capaç de comparar les velocitats de pujada i baixada dels dos participants del canal WebRTC arribant a un consens que proporcioni el millor funcionament.

En SRTM aquest consens es fa de la següent manera:

-L'usuari 1 envia els amples de banda de pujada de cada medi a l'usuari 2 abans de crear l'SDP, per altre banda l'usuari 2 fa el mateix.

-Quan els dos usuaris tenen els amples de banda de pujada de l'altre els comparen amb els seus amples de banda de baixada escollint els menors dels dos. D'aquesta manera no obligarem mai a l'altre usuari a enviar per sobre de la seva capacitat ni a nosaltres a rebre per sobre de la nostre.

-Un cop escollits els valors d'ample de banda que s'usaran creen els objectes SDP i se'ls intercanvien.

D'aquesta manera que s'ha dit, l'usuari 2 mai obligarà a l'usuari 1 a enviar una quantitat de dades per segon major a la que ell volia, ja que abans de crear l'SDP s'ha tingut en compte la seva capacitat de pujada i no només la capacitat de baixada de l'usuari 2, que al tenir probablement una xarxa asimètrica, sigui molt major que la capacitat de pujada de l'usuari 1.

Per constatar això es va fer una proba entre 2 usuaris amb xarxes que tenien velocitats de transmissió molt diferents i es va veure que no ni havia prou amb limitar les velocitats de transmissió locals per tal de que les limitacions tinguessin efecte en l'enviament del vídeo (allà on la velocitat de transmissió és menor). Per això és necessari que els dos usuaris s'intercanviïn l'ample de banda de pujada abans de crear els SDP.

Els objectes SDP no només es poden modificar un cop creem la connexió *WebRTC*, un cop establert el canal WebRTC també es poden canviar. En aquest aspecte però cal tenir cura amb que és el que es canvia. La llibreria ASWRTC que s'ha implementat té la capacitat de renegociar les velocitats de transmissió si es veu que les actuals no estan donant resultats. D'aquesta manera si el programador que usa la llibreria no li passa correctament les velocitats de transmissió el sistema sigui capaç de sobreposar-se i reassignar unes VDT que, encara que no siguin les més ajustades, permetin l'intercanvi de flux de manera fluida.

Per detectar que les VDT actuals estan perjudicant l'estabilitat de la videoconferència, ASWRTC utilitza la llibreria ***getStats*** proporcionada també per WebRTC. Aquesta llibreria li permet obtenir el numero de paquets que s'estan perden respecte el total, i si aquest numero de paquets perduts és molt elevat llavors es disminueixen les VDT.

Per renegociar les VDT el que fa la API ASWRTC és, quan detecta a través de l'API *getStats* que el nombre de paquets perduts (***packetsLost***) és major al nombre de paquets rebuts (***packetsReceived***) llavors es modifica l'SDP local indicant dos noves VDT pels medis d'àudio i vídeo més baixes que els que fins ara s'estaven utilitzant, un cop actualitzat l'SDP local amb el mètode *setLocalDescription()* s'envia a l'altre usuari perquè modifiqui l'objecte SDP remot (que és el nostre SDP local) amb el *setRemoteDescription()*.

Aquest fet redueix la quantitat d'informació que ens ha d'enviar l'usuari remot, perdent així qualitat en els *streams* que envia però guanyant fluïdesa respecte aquests.

WebRTC, té la variable interna paquetsEsperats, que és la suma de paquets perduts més els rebuts [45]. D'aquesta manera, la plataforma ASWRTC considera que s'està tenint problemes d'estabilitat quant la suma de paquets perduts és més gran que la de rebuts, és a dir, si més del 50% de paquets esperats d'un *MediaStreamTrack* que s'està rebent es perden, llavors es baixen les velocitats de transmissió del SDP Local per no exigir tanta quantitat d'informació a l'usuari remot, ja que per voler satisfer les necessitats de l'usuari local està perdent la majoria dels paquets en cues d'espera.

Quan s'indica a l'objecte SDP una limitació de velocitats de transmissió de baixada d'un *MediastremTrack* el que es fa és reduir la qualitat del flux que es rep. Per exemple, en un objecte *MediastremTrack* de vídeo, si es recupera el vídeo amb una resolució de *640x480* si es limita la VDT llavors se li està indicant al còdec encarregat de comprimir i enviar la seqüència

d'imatges que no ha de codificar tots els píxels de totes les imatges que veu i enviar-los, sinó que ha de codificar i enviar el numero de píxels per imatge que tingui temps de fer amb el ***bitRate*** (velocitat de transmissió) que se li ha establert. Aquest fet pot fer que l'usuari que envia el vídeo, tot i que la seva càmera li recuperi *640x480* píxels per segon enviï la resolució que el bitRate i còdec li limiten.

Totes aquestes dades de l'objecte SDP que no afecten al canvi de ICE Candidats o qualsevol altre cas en que s'alterin els l'origen i destí dels fluxos enviats poden ser modificades un cop la conversa ja s'ha inicialitzat (el canal WebRTC ja està establert). Altrament, si es volen modificar dades d'orígens i destins del fluxos multimèdia es necessari tornar a fer l'intercanvi de candidats i s'ha de tancar el *PeerConnection* i tornar a crear.

Tot i que fins ara només s'ha parlat de fluxos multimèdia i de senyals d'àudio i vídeo, **en la videoconferència, a part d'usar les llibreries de MediaStream i PeerConnection també es proporciona un xat P2P implementat a partir de l'API de DataChannel**, en el xat però, no es limiten les velocitats de transmissió ja que es controla la quantitat d'enviament de dades, obligant a que missatges amb més de 10000 caràcters s'hagin de particionar i ser enviats per parts (això s'ha implementat gràcies a la creació d'objectes *FileReader*) cada un cert interval de temps. D'aquesta manera es permet mentre s'envien grans fitxers, poder seguir enviant missatges més curts (per exemple de text) entre mig a través del *DataChannel*.

ÀMBIT D'UTILITZACIÓ DE NODE.JS

NodeJS s'ha usat com a servidor de l'aplicació (Signaling Controller Server). Oferint una API Rest que permetés al client web poder interactuar amb la BD i també per poder establir comunicacions d'igual a igual entre el servidor i navegador en Socket.IO, sense la necessitat d'usar llibreries WebSocket diferents alhora d'implementar-los.

Així doncs en el servidor **SCS** tenim una API Rest que es comunica amb la base de dades, més una interfície de missatges en Socket.IO creada inicialment per permetre el protocol de signaling de WebRTC.

Per poder usar aquests serveis en el servidor s'han hagut d'expandir les funcionalitats inicials de Node amb la importació de mòduls. Els mòduls importats són:

-HTTPS [14]

Per poder implementar i donar servei als tipus de crides GET, POST, PUT i DELETE per obtenir, afegir, modificar i esborrar dades del servidor des del client.

-FS [15]

Per poder llegir i modificar altres fitxers del sistema. Gràcies aquest mòdul es pot importar qualsevol fitxer, podent així afegir els certificats SSL que necessita el mòdul HTTPS per treballar de manera segura, i permetent estructurar el codi en fitxers d'una manera molt senzilla.

-SOCKET.IO [16]

Aquest mòdul permet afegir al servidor la capacitat de tenir connexions persistents amb els navegadors. Un cop afegit el mòdul es poden crear **n** seccions (namespaces) (que són com classes) i oferir a cada *namespace* una sèrie de mètodes que rebin com a paràmetres dades dels clients i els i retornin respostes.

-EXPRESS [17]

Al inici no s'esperava usar aquesta funcionalitat, però degut als problemes de cross-domain que no vaig ser capaç de solucionar amb el mòdul *node-static*, es va decidir incorporar aquest framework que va donar molta facilitat per solucionar el problema inicial a les peticions HTTP provinents d'altres dominis.

-BODY-PARSER [18]

Assisteix a l'entorn de treball *express* per millorar la comunicació amb el client, analitzant sintàcticament (parcejant) l'entrada de les peticions HTTP a, per exemple, JSON, per així fer entendibles els objectes enviats pel client.

-ASYNC [19]

Al ser NodeJS un servidor asíncron, quan es vol fer per exemple una seqüència de peticions concurrents a la base de dades és impossible saber quan totes les peticions han acabat; fent així que si vols executar **x** codi un cop estiguin totes les peticions fetes tinguis un problema.

El mòdul ASYNC s'usa a SRMT per tal de solucionar casos en que després d'haver executat una funció concurrent necessitem reprendre l'execució. Per ser més exactes, s'usa aquest mòdul quan dintre d'un procés iteratiu es crida una funció concurrent, i se solen usar les funcions ***each*** (per fer iteracions que tenen processos concurrents i executar una funció un cop es té el resultat de tots els processos), ***waterfall*** (per executar varies funcions concurrents en ordre) i finalment ***forEachOf***, que fa el mateix que ***each*** però itera vectors d'objectes.

-BCRYPT [20]

Al tractar l'aplicació amb dades de caràcter personal és necessari guardar les contrasenyes de manera xifrada. Aquest mòdul ofereix una manera molt fàcil de fer-ho, és per això que s'usa.

El mòdul , tal com s'indica l'enllaç ofereix un mètode per generar un resum hash a partir de la clau i un altre per comparar la clar amb un resum hash. Quan un usuari es registra envia la seva contrasenya al servidor, aquest genera a partir de la contrasenya, usant la funció computacionalment no invertible BCRYPT.***hash***, un resum únic. El que es guarda al servidor és aquest resum.

Gràcies al fet de que amb la mateixa contrasenya sempre generarem la mateixa funció hash, quan un usuari s'identifica, el que es fa per comprovar que la contrasenya sigui correcte, és generar un resum hash de nou i comprovar que és el mateix que estava guardat a la base de dades; d'això se'n encarrega la funció BCRYPT.***compare***, si el resum és el mateix l'usuari accedeix, sinó no.

ÀMBIT D'UTILITZACIÓ DE SOCKET.IO

Socket.IO inicialment estava pensat per ser usat en la implementació del protocol de senyalització de WebRTC. WebRTC necessita un servidor reflexiu per a que els 2 usuaris participants en un PeerConnection puguin intercanviar-se almenys els objectes SDP; per proporcionar aquesta funcionalitat a WebRTC i tenint en compte el llibre ***Real Time Communications with WebRTC***, en que posa un exemple de protocol d'inicialització amb un servidor Node i Socket.IO, es va escollir Node.JS com a servidor reflexiu i la llibreria Socket.IO com a canal.

A partir d'aquí és van estructurar els missatges de WebRTC en un *namespace* (un namespace és com una classe sense mètodes, sol contenir els mètodes d'un determinat tipus o que tenen una determinada utilitat) i es va crear un nou *namespace* per poder utilitzar Socket.IO per a enviar tot tipus de dades entre client i servidor (i viseversa), com per exemple, missatges per controlar l'estat dels usuaris, proporcionar un xat en temps real als usuaris i altres.

La creació de *namespaces* està pensada perquè quan un usuari entra a l'aplicació se li obra un *Socket* per enviar-se missatges en temps real amb el servidor, llavors quan entra en una videoconferència se li obre un altre *Socket* pel protocol de senyalització. La separació en *namespaces* està pensada sobre l'idea de que en un futur, els dos *namespaces* es puguin situar en servidors diferents per tal de no carregar un únic servidor amb totes les connexions Socket.IO.

D'aquesta manera SOcke.IO s'usa tant en el protocol de senyalització de WebRTC com per a canal bidireccional per a gestionar el perfil dels usuaris en temps real.

Socket.IO a la part servidora

Pel que fa el servidor, que és on s'estructuren els mètodes en *namespaces*. Cal declarar els *namespaces* de la següent manera:

***io** és la variable associada al mòdul "socket.io"*

```
io.of('/nomNameSpace').on('connection',function(socket){
  //Funcions Socket.on (socket.on...)
});
```

Un cop declarat el *namespace* es pot ficar a dintre seu tots els mètodes que tingui disponible el client per a fer peticions al servidor. Des d'aquests mètodes es pot usar qualsevol variable global, permetent així que des dels mètodes Socket.IO es pugui també fer crides a la BD.

Intercanvi de missatges entre servidor i client

Un cop el servidor ha creat el protocol de mètodes, el client que és connecti al *namespace* els podrà utilitzar per enviar-li missatges (però en la part client també es poden implementar mètodes perquè el servidor els cridi). Per establir una connexió *socket* amb el servidor, el client ha de cridar el mètode *io.connect* de la llibreria Socket.IO pel navegador:

```
var socket=io.connect('@IP:port del servidor,'/ nomNameSpace');
```

A partir d'aquí tant el client com el servidor usaran ***socket.emit*** per enviar missatges i ***socket.on*** per rebre'ls. És a dir, si volem posar una funció tant a la part client com a la servidora posem:

```
socket.on('nomFunció, function(paràmetres) {
  //Codi de la funció
});
```

Aquesta funció pot ser cridada enviant un esdeveniment socket.emit de la següent manera:

```
socket.emit('nomFunció, paràmetres);
```

Respecte els esdeveniments *socket.emit*, la única diferència entre client i servidor és que el client només pot cridar mètodes *socket.on* del servidor, mentre que el servidor, pot cridar els mètodes *socket.on* de qualsevol client, independentment que sigui el que li ha enviat l'esdeveniment o no.

En el protocol de senyalització de WebRCT per exemple, s'usa el mètode *socket.broadcast.to("idConversa").emit('*nomFunció,paràmetres*); per cridar el mètode* nomFunció de la part client, per tots els clients del grup *idConversa* menys el client que ens ha enviat l'esdeveniment (sí volguéssim incloure el client que ha enviat l'event usaríem *socket.broadcast.in* en comptes de *socket.broadcast.to*).

Com s'ha dit en aquest últim paràgraf en Socket.IO es poden fer grups d'usuaris, per poder enviar així esdeveniments cap a certs grups. Això només s'usa en la part de senyalització de WebRTC, fent grups entre tots els usuaris que participen en una videoconferència determinada.

6. Requisits del sistema

El problema inicial al qual s'enfrontava el projecte era l'idea de, havent vist el sistema VITAM, ser capaços de proporcionar una llibreria basada en *WebRTC* que oferís al igual que LICODE, una interfície que permetés la fàcil implementació d'un sistema de multi conferències.

A diferencia de LICODE aquesta llibreria està destinada a connexions P2P i ha de proporcionar un alt control dels fluxos multimèdia que s'intercanvien els usuaris de WebRTC, permetent observar en especial la velocitat de transmissió consumida per cada medi dels usuaris.

Amés d'això per resoldre el problema d'escalabilitat a la part del client, degut al fet de que per cada usuari que hi ha a la videoconferència cal enviar-li els nostres fluxos i rebre els seus, la API ASWRTC ha de permetre la regulació dels amples de banda del client, fent que el client no envií ni rebi més dades per segon de les que li permet la seva velocitat de transmissió de xarxa.

A partir d'aquesta llibreria, que soluciona el problema que té *WebRTC* en conferències múltiples (no té en compte la velocitat de transmissió de xarxa de l'usuari fent que els fluxos es sobresaturin), s'ha de permetre que un programador pugui incorporar un sistema de videoconferències sense la necessitat de preocupar-se de com modelar-ho.

Per altre banda, SRTM ha de proporcionar una interfície còmode perquè els usuaris puguin gestionar el seu perfil, agregar usuaris i xatejar amb ells. Pel que fa a la part de videoconferència, s'ha d'oferir una interfície que proporcioni a l'usuari el control d'enviament de les senyals d'àudio i vídeo, la possibilitat de gravació d'aquestes senyals i al igual que en la part del perfil, un xat que permeti el intercanvi de missatges de text, imatges i fitxers. La diferència entre el xat de la videoconferència i el del perfil, és que el del perfil ha de ser persistent i el de la videoconferència no.

A partir d'aquesta idea inicial ara es passarà a la fase d'anàlisi dels requisits on s'especifiquen els requisits que ha de tenir el sistema, els requisits mostrats són tots de cares a la interacció amb l'usuari, per tant, no es mostren els requisits de la llibreria que s'ha construït com a pont entre les API's de WebRTC i la part d'interfície de l'usuari (ASWRTC).

6.1. Requisits funcionals

Aquest apartat conté tots els requisits funcionals que ha de tenir el nostre sistema de cares a l'usuari, tant en la part de gestió de l'usuari i converses, com un cop dins d'una videoconferència.

Aquí es mostra un resum de tots els requisits i en la secció 1 [Extensió dels requisits funcionals] de l'apartat d'annexes hi ha una versió ampliada d'aquests.

Els requisits funcionals es poden agrupar en tres grans grups, els destinats al registre i identificació de l'usuari, els destinats per a la gestió del perfil i converses i els necessaris per a la gestió de videoconferències.

Requisits pel registre i identificació

- Registrar usuari.

  Permetre a un usuari registrar-se a l'aplicació Web per usar-la.

- Identificar usuari.

  Per l'accés al seu perfil per els usuaris registrats.

Requisits per a la gestió del perfil i contactes

- Modificar perfil

  Permetre el canvi de les dades personals que identifiquen un usuari.

- Gestionar usuaris

  Gestionar tan la busqueda de nous contactes com l'opció d'envia'ls-hi peticions.

- Gestionar peticions conversa

  Informar a l'usuari de les peticions pendents de resposta i permetre-li respondre les peticions que li han enviat.

- Gestionar amics

  Llista les amistats de l'usuari perquè pugui envia'ls-hi peticions de converses múltiples .

- Gestionar converses

  Permetre la gestió de les converses en les que participa l'usuari, permetent-li esborrar-les o obrir-les per tal d'enviar dades o realitzar videoconferències.

- Mostrar trucades actives

  Informar a l'usuari de les converses en que té una videoconferència oberta.

- Gestionar amples de banda

  El sistema és capaç d'avaluar les velocitats de transmissió de pujada i baixada disponibles a la xarxa en que està connectat l'usuari.

- Gestionar connectivitats

  Donar a l'usuari la informació de quins amics estan en línia i quins no en un moment determinat.

Requisits per a la gestió de videoconferències

- Gestionar usuari local

  Donar la possibilitat que l'usuari pugui gestionar els fluxos multimèdia que envia als demés podent parar-los/engegar-los i gravar-los.

- Gestionar usuaris remots

  Permetre gravar els fluxos multimèdia provinents d'altres usuaris, parar-los i reprendre'ls.

- Gestionar xat

  Proporcionar un xat pel qual els usuaris presents a la videoconferència es puguin intrecanviar dades.

-Gestionar VDT

  Disminuir les velocitats de transmissió si la conversa no és estable per estabilitzar-la.

Independentment a aquests requisits per a la interfície de l'usuari el sistema ha de tenir, pensant en la llibreria sobre WebRTC construïda els següents requisits:

- Proporcionar una API per a la fàcil obtenció de les estadístiques d'una conversa. Aquesta API ha de permetre recuperar les velocitats de transmissió, ICE candidates i saber el nombre de paquets perduts durant la videoconferència.

- Proporcionar una API per a la creació de multi conferencies escalables, oferint la possibilitat de que es puguin regular les VDT des de fora l'API i també la API ha de ser capaç d'autorregular-se les seves velocitats de transmissió.

Els requisits sobre la persistència de les dades són exposats juntament amb la fase d'anàlisi de la Base de dades.

6.2.Requisits no funcionals

- És una plataforma Web.

  En un futur l'aplicació pot passar a ser multi plataforma i donar servei a IOS i Android, actualment però, pel projecte, només s'espera donar servei Web

-Programat en JavaScript, HTML i CSS.

- Interfície gràfica de fàcil comprensió.

  S'espera que l'usuari pugui usar l'aplicació sense la necessitat de cap manual d'ajuda.

-Servidor amb base de dades.

  S'ha de proporcionar un servidor amb base de dades per tal de poder guardar les dades dels usuaris de manera persistent.

-El servidor ha de respondre peticions HTTP.

  S'ha de proporcionar un servei per a que se li puguin fer peticions HTTP.

-El servidor ha de permetre executar PHP.

  S'ha de tenir un servidor php per tal de poder passar paràmetres entre pàgines i fer comprovacions dinàmiques en els arxius html.

-Canals bidireccionals entre client i servidor.

  Hi ha d'haver un protocol de missatges bidireccionals per el procés de Senyalització previ a establir una connexió WebRTC i el xat entre usuaris en temps real.

-Permetre la connexió d'usuaris des de qualsevol tipus de xarxa.

  El servidor ha d'estar en una @IP pública i són necessaris els servidors STUN i TURN per, alhora de les videoconferències, comunicar usuaris que es troben en xarxes privades.

- Capacitat de processar gran quantitat de peticions.

  El servidor ha d'estar preparat per a respondre una gran quantitat de peticions dels clients sobre la base de dades.

- Suportar una gran quantitat de connexions TCP.

  Degut al fet de que s'usa socket.IO per a que el servidor pugui enviar events als clients sense que aquests hagin de demana'ls-hi abans, és necessari que el servidor estigui preparat per a poder tenir la major quantitat possible de socket en línia, ja que per cada usuari connectat es necessita un socket obert entre client i servidor.

- Guardar les dades referents als usuaris de manera encriptada.

  Per tal de complir les normes jurídiques de la llei LOPD cal guardar les dades de manera que cap persona no desitjada pugui llegir-lo.

- Enviar les dades xifrades a través de la xarxa.

  Funcionalitat per al compliment de la llei LOPD referent al nivell de seguretat alt.

- **Es dona suport per Chrome i Firefox**.

  La resta de navegadors no s'han provat, per això no se sap com respondrà l'aplicació. En principi ha de funcionar en qualsevol navegador excepte IntrnetExplorer i Safari, que són els únics navegadors que no tenen implementades les funcionalitats de la llibreria de WebRTC.

-En Firefox no es dona suport en el control de les velocitats de transmissió.

  En Firefox, la modificació dels objectes SDP no té afecte, degut això no hi ha manera de limitar les velocitats de transmissió dels usuaris.

6.3. Diagrama de casos d'ús

A partir dels requisits funcionals, com a segona part de la fase de requeriments es mostren els diagrames de casos d'ús.

Gràcies a aquest diagrama (modelat en UML) es mostren gràficament les necessitats del sistema en base als events de negoci, persones o dispositius que inicien els esdeveniments i la resposta del sistema sobre els events.

Per mostrar els casos d'ús del sistema s'ha fet un diagrama per a cada pàgina web, amb la idea de que l'usuari que activa els events del negoci de cada diagrama té un rol diferent.

El primer diagrama es per les funcionalitats que té un usuari no autentificat, el segon diagrama es mostren les funcionalitats en base a la pàgina principal de gestió de dades i en el tercer esquema es veuen les funcionalitats que ha de proporcionar el sistema a un usuari que està fent una videoconferència.

Diagrama casos d'ús per a la pàgina inicial:

A la pàgina https://kidd.udg.edu/ hi han tres botons i només es mostren dos events de negoci. Això es degut a que el primer boto únicament et redirigeix a la pàgina i no s'ha tingut en compte.

L'usuari que pot activar els casos d'ús és un usuari que no s'ha autentificat encara a l'aplicació, sigui perquè encara no i forma part o perquè, tot i haver-se registrat no ha validat el seu usuari i contrasenya. Un usuari no registrat, tot i que pot executar la primera funcionalitat (identificar usuari) només n'obtindrà avisos d'accessos no autoritzats fins que al cap de tres intents se li negarà la possibilitat de tornar a executar el primer cas d'ús.

Sigui quin sigui el cas d'ús que s'executa, el resultat de tots dos events de negoci és accedir a la pàgina personalitzada, on podrà gestionar el seu propi perfil, agregar amics i xatejar amb ells.

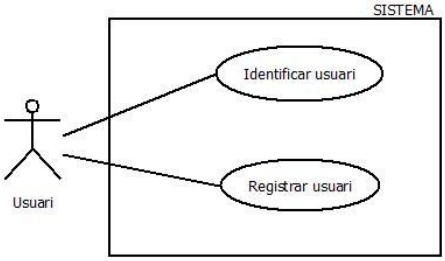


Diagrama casos d'ús per a la pàgina de gestió del perfil:

En aquest segon diagrama si accedeix només si hem passat per a la pàgina d'autentificació satisfactòriament. En aquest cas i intervenen dos actors, el segon actor, el sistema, llança els events de negoci que malgrat tenir repercussió amb la interfície no són llançats per a cap usuari que interactua amb ell, sinó que és el mateix sistema que els genera. Per altre banda l'actor *Usuari identificat* representa un usuari que ha entrat a l'aplicació i interactua amb el nostre sistema a través de la interfície, així doncs, l'usuari del diagrama anterior ha canviat de rol i passa a tenir les funcionalitats de l'actor *Usuari identificat*.

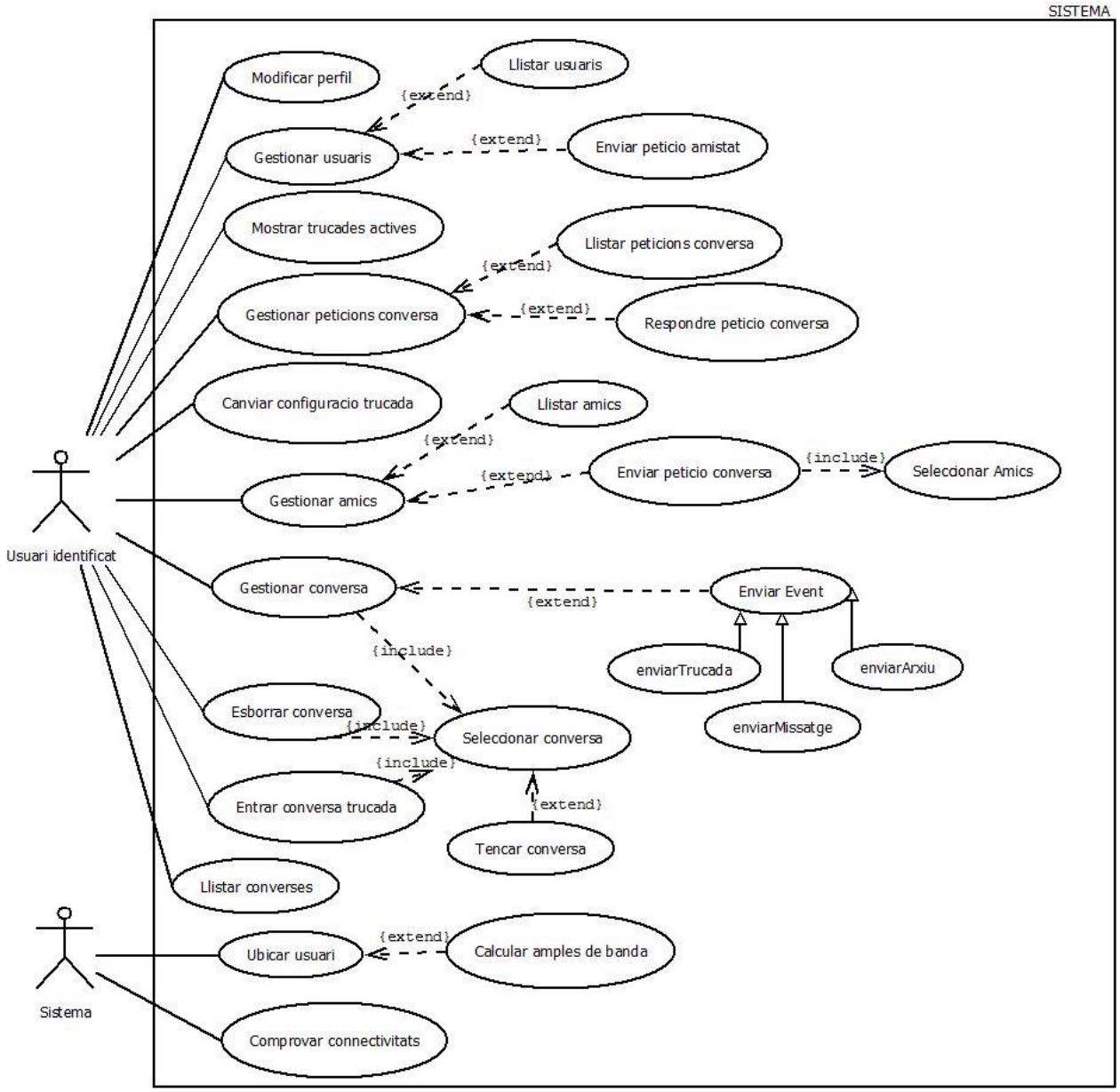


Diagrama casos d'ús per a la pàgina de videoconferències:

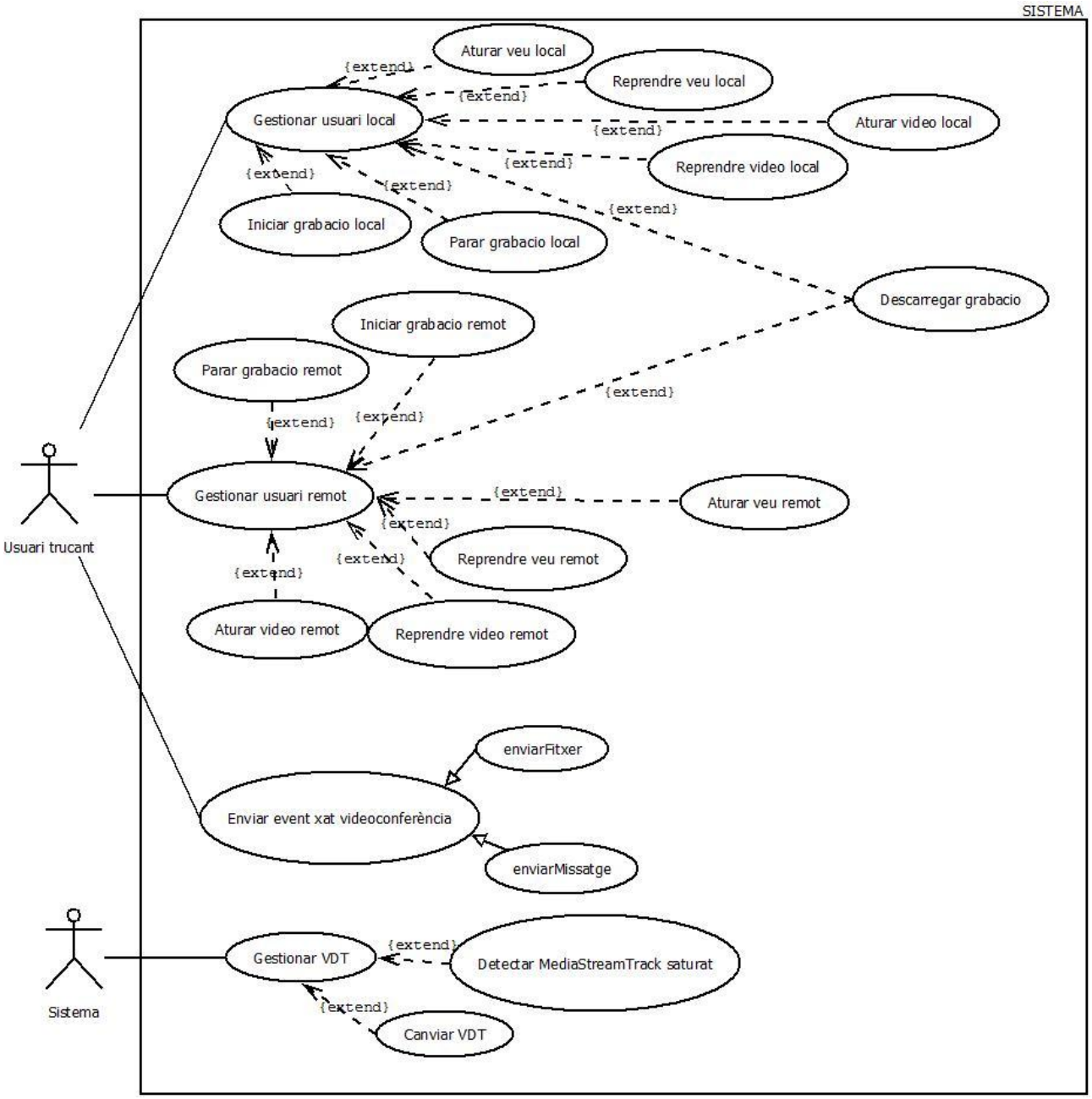
El diagrama mostra les funcionalitats que ha d'oferir la interfície a un usuari que utilitzi una videoconferència per comunicar-se amb els demés usuaris d'una conversa. En aquesta interfície si accedeix quan, des del xat d'una conversa persistent enviem una trucada i ens l'accepten, o quan rebem una trucada i l'acceptem.

A l'actor *Usuari trucant* (usuari dins d'una videoconferència) se li ofereixen tot de possibilitats relacionades amb el control de fluxos multimèdia, amés d'un xat molt igual al de l'anterior pàgina però aquest cop els missatges i altres dades enviades no són persistents.

Així com les anteriors funcionalitats necessitaven enviar i rebre dades del servidor SCS, tots aquestes casos d'ús de l'usuari trucant són independents al servidor, de fet es pot parar el servidor i les funcionalitats de l'usuari trucant seguiran totes en funcionament, cosa que no passa en els altres diagrames.

Per la funcionalitat ***Canviar VDT*** que pot llançar l'actor sistema si que és necessari fer passar el nou SDP que indica el canvi de velocitats de transmissió pel servidor. Aquestes funcionalitats del sistema són les que permeten estabilitzar *PeerConnections* inestables degut a la insuficiència de VDT a la xarxa.

Amb això aconseguim sobrecarregar el servidor el mínim possible.



6.4. Diagrames d'activitats

En aquest apartat, per acabar la fase de requeriments, es mostra, per cada cas d'ús, una gràfica del flux d'activitats involucrades (ordre en que s'executen les activitats i com depenen unes de les altres). Al igual que en l'apartat 6.3 es mostraran els diagrames d'activitat dels cas d'usos separats per pàgines.

DIAGRAMES D'ACTIVITAT DE LA PÀGINA D'INICI

En aquesta pàgina tenim dos caos d'ús tal com es mostra en la imatge 15:

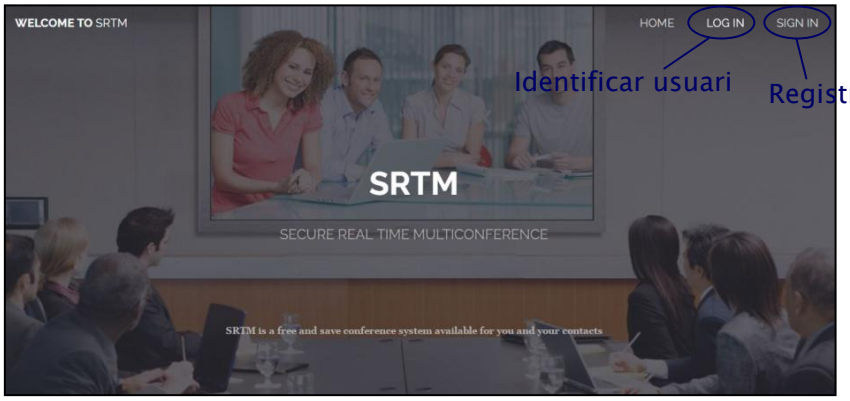
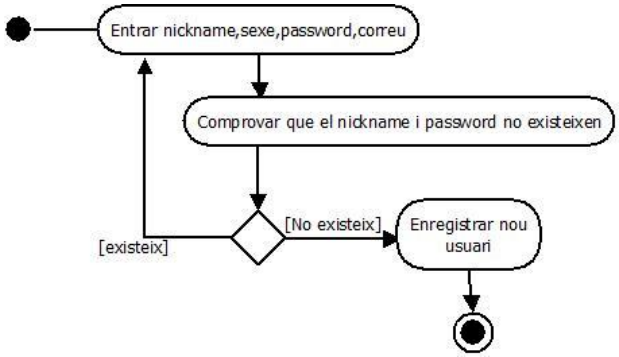


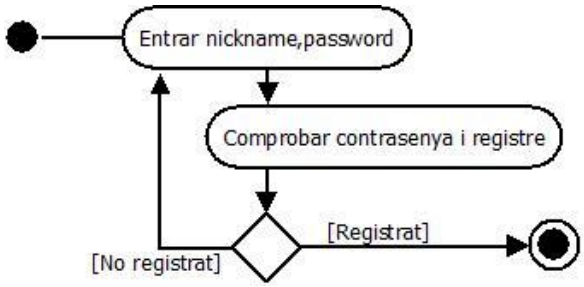
Figura 15

Cas d'ús registrar usuari:



Quan l'usuari emplena el formulari i envia una petició de registre al servidor, el servidor ha de comprovar que els camps únics no existeixin a la BD. Si tot està correcte l'usuari queda registra al sistema i passa a la pàgina d'inici.

Cas d'ús identificar usuari:



Si un usuari ja registrat s'autentifica per entrar de nou a la seva pàgina personalitzada ha d'entrar nom i contrasenya. Al fer-ho s'envia la petició al servidor, que comprova que existeix un usuari amb aquell nom i contrasenya. Si tot és correcte s'entra a la pàgina personalitzada, sinó s'ha de tornar a validar el qüestionari. Si un usuari s'equivoca més de tres cops no se li permet entrar a la pàgina.

DIAGRAMES D'ACTIVITAT DE LA PÀGINA DE PERFIL I GESTIÓ DE CONVERSES

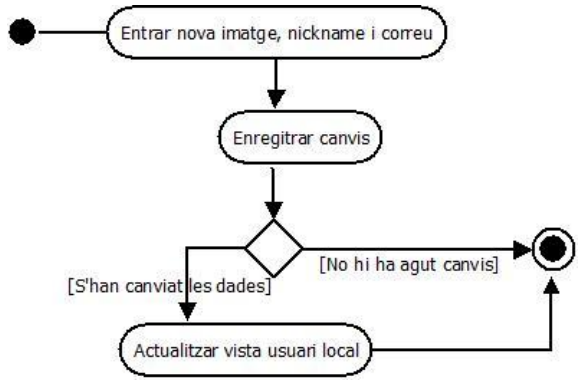
En aquest apartat és on hi ha el major nombre de casos d'ús:

Figura 16

En la figura 16 es mostra una mica per sobre la major part dels esdeveniments que pot llençar l'usuari, en al captura que s'ha fet falten només els esdeveniments per enviar i respondre peticions que s'obren quan es fa clic sobre un usuari remot dels llistats de amics, usuaris o peticions conversa.

Seguidament es mostra la seqüència d'activitats que s'executen en cada cas d'ús al accionar el cas d'ús relacionat amb l' esdeveniment:

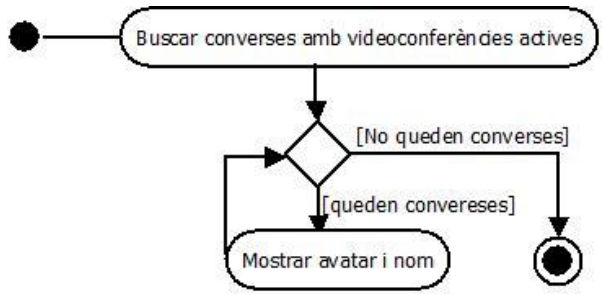
Cas d'ús modificar perfil:



Quan un usuari vol canviar les seves dades de perfil s'executa aquest diagrama. L'usuari pot canviar la imatge, nom o el correu.

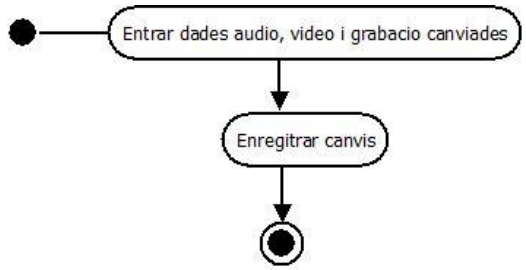
En el cas de canviar el nom o correu serà necessari només l'actualització de la base de dades, però per les imatges, s'haurà de pujar la nova foto de perfil al servidor i llavors enviar la nova URL a la BD. Quan un usuari actualitza la foto de perfil automàticament es borra la foto que tenia anteriorment.

Cas d'ús mostrar converses actives:



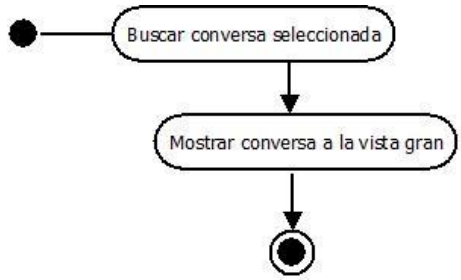
Quan l'usuari prem el botó *ACTIVE CALLS* se li mostra un llistat de totes converses en que té la videoconferència oberta. A través d'aquest llistat, en un futur, es té previst incorporar l'opció de sortir de la videoconferència des de la pàgina de gestió de converses.

Cas d'ús canviar configuració trucada:



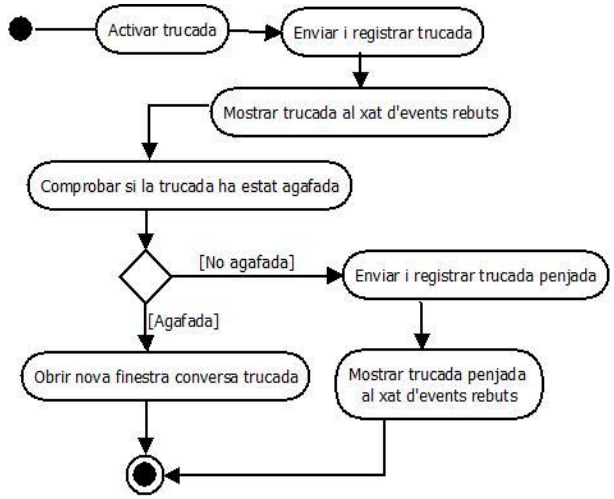
L'usuari pot escollir si usarà el micròfon, la càmera o si voldrà gravar i ser gravat. Quan es detecta un canvi en les característiques d'entrada es canvia l'estat del formulari i s'actualitza la BD amb les noves característiques d'entrada a la trucada.

Cas d'ús seleccionar conversa:



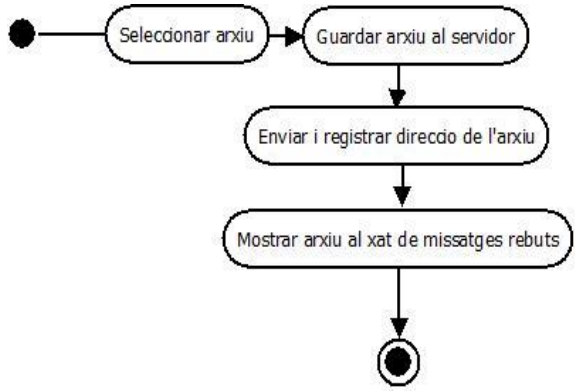
A partir de la llista de converses en les que participa l'actor usuari identificat en selecciona una i l'obra. Aquest fet implica obtenir els últims events enviats a la conversa (trucades, missatges, imatges...).

Cas d'ús enviar trucada:



Quan l'actor realitza una trucada sobre una conversa per obrir una videoconferència tots els usuaris que estan en línia i que participen a la conversa reben la trucada. Si cap dels usuaris accepta la trucada llavors no es realitza la videoconferència. Però si almenys un usuari accepta la trucada ja s'obra la videoconferència. En aquest cas tots els usuaris que havien rebutjat la trucada se'ls i esborra la possibilitat de realitzar trucades per aquella conversa i se'ls i ofereix un botó per entrar directament a la videoconferència.

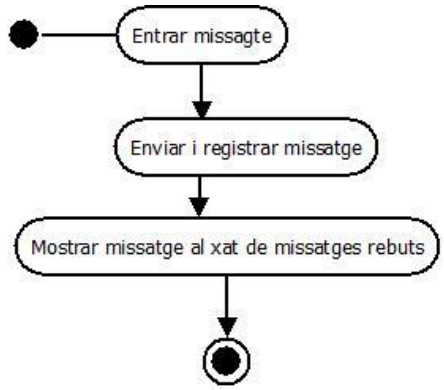
Cas d'ús enviar arxiu:



Quan un usuari envia un arxiu sobre una conversa el que es fa és pujar l'arxiu al servidor i enviar als demés usuaris la URL única per a que puguin descarregar l'arxiu del servidor.

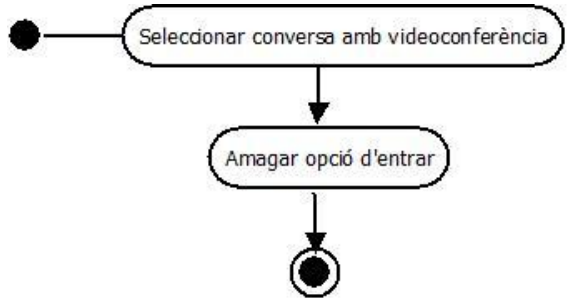
A la BD també es guarda aquesta URL de manera persistent.

Cas d'ús enviar missatge:



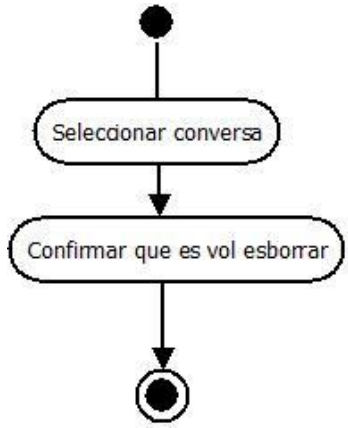
L'actor usuari identificat envia un missatge per a la conversa que té seleccionada. Un cop el missatge arriba als demés usuaris a ell també se li mostra en el xat.

Cas d'ús entrar conversa trucada:



En el cas d'arribar tard a una videoconferència activa o després d'haver-la rebutjat i que s'obrís, a l'usuari se li dona aquesta funcionalitat per que pugui entrar a una videoconferència que està en curs.

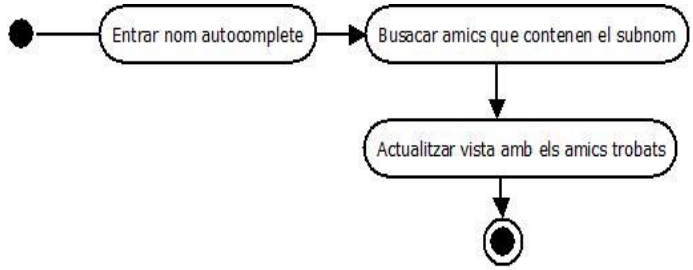
Cas d'ús esborrar conversa:



Aquesta funcionalitat, un cop l'usuari confirma que vol esborrar la conversa, envia un missatges a la BD i a la resta d'usuaris de la conversa que estan en línia per tal de que esborrin definitivament la conversa.

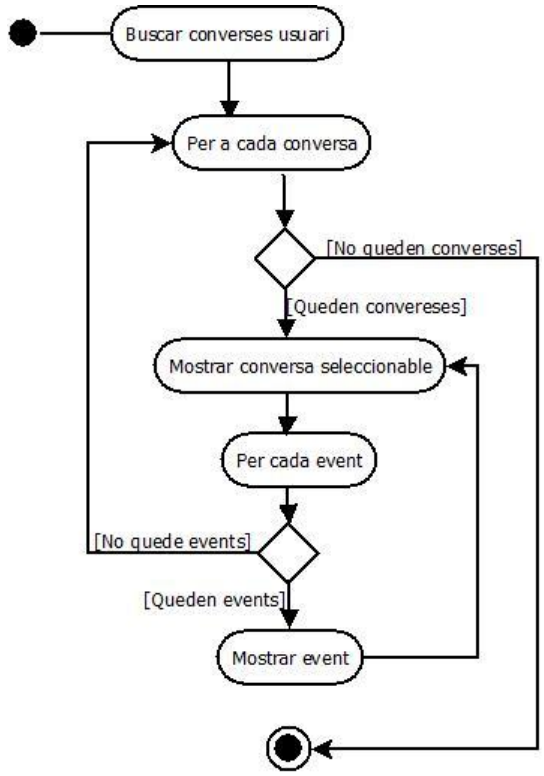
Al esborrar la conversa es neteja tot l'historial de missatges relacionada amb ella.

Cas d'ús llistar amics:



Quan l'usuari entra a aquesta segona pàgina se li mostren tots els amics en una llista (la majoria però queden ocultats). Per evitar que l'usuari hagi d'anar buscant entre tots els amics un amic en concret, se li dona un buscador que permet entrar part del nom de l'amic i llavors el sistema selecciona només els amics que compleixen amb el patró per tal de reduir la llista.

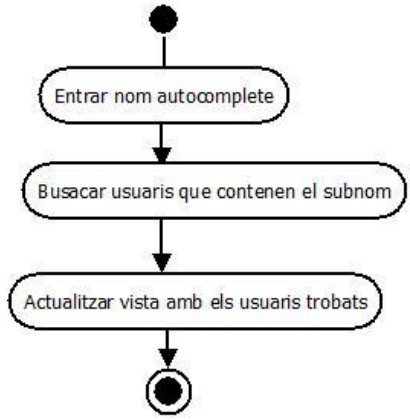
Cas d'ús llistar converses:



Quan l'usuari entra a aquesta segona pàgina se li mostren totes les converses en les que participa. Amés per a cada conversa es recupera també el conjunt dels esdeveniments més actuals.

Encara que l'usuari no seleccioni la conversa ja es descarreguen del servidor els últims esdeveniments per a que quan la seleccioni se li mostrin els missatges rebuts instantàniament.

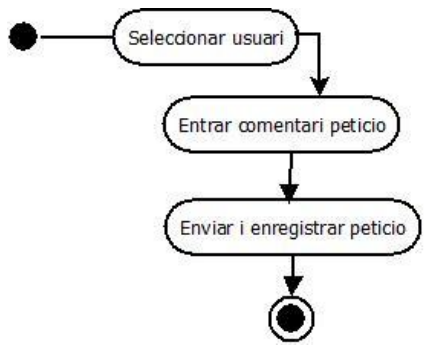
Cas d'ús llistar usuaris:



A l'actor se li mostra inicialment un llistat dels vint primers usuaris recuperats a la base de dades.

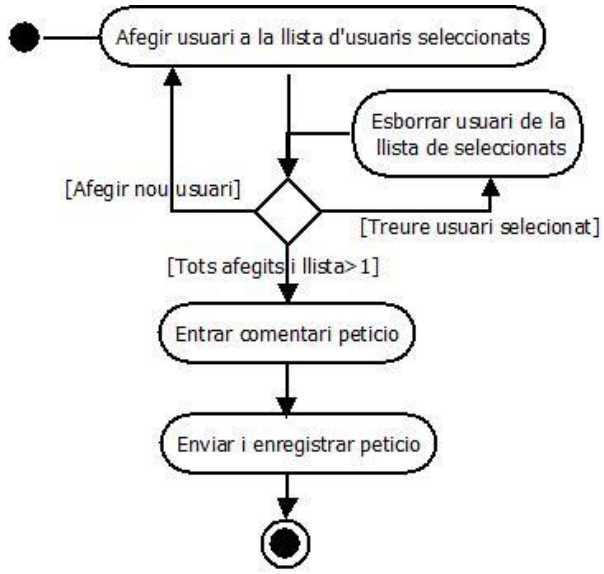
Però probablement l'usuari que li interessa a l'actor no estigui en aquesta llista, per això, se li dona un buscador que cada cop que passa 1 segon després de detectar un canvi obté 10 usuaris nous de la base de dades en que el seu nom compleix el patró del nom entrat.

Cas d'ús enviar petició amistat:



Quan l'actor envia una petició a només un altre usuari amb l'objectiu de que els dos puguin realitzar una conversa en un futur aquesta petició es guarda a la BD i arriba a l'altre usuari. En aquest moment els dos usuaris actualitzen la vista de peticions i mostren la petició. A l'actor que a llançat l'event li surt que la petició està en espera de resposta. A l'altre usuari en canvi li surt la petició amb l'opció d'acceptar o rebutjar-la

Cas d'ús enviar petició conversa:

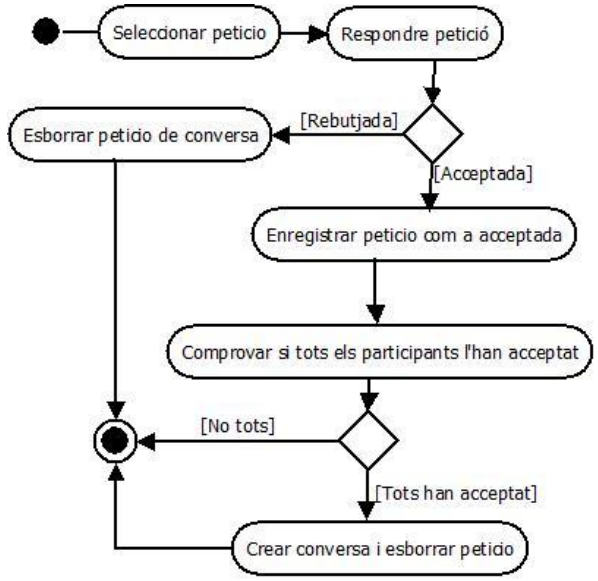


En aquest cas l'actor va a la llista d'amics i selecciona uns quants amics per crear una conversa múltiple.

L'actor pot anar seleccionant o traient amics i se li mostren els amics que actualment té seleccionats en un llista a sota de la vista.

Quan considera que hi ha tots els amics als qui volia enviar la petició l'actor llança la petició i es segueix el procés explicat en l'anterior cas d'ús però per múltiples usuaris.

Cas d'ús respondre petició conversa:

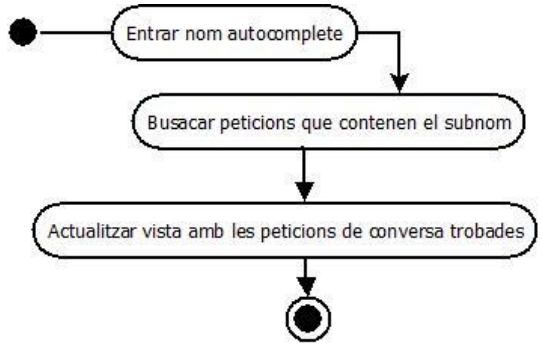


Quan l'actor rep una petició de conversa (ja sigui individual o múltiple) i la respon, té l'opció d'acceptar-la o rebutjar-la. Si es rebutja la petició s'esborra i l'actor ja no en sabrà mai més res d'aquella conversa.

Si s'accepta, si la petició era individual ja es passa a crear la conversa i es mostra instantàniament, sinó, si la petició era per a variïs usuaris, és possible que calgui esperar a que tots els usuaris l'acceptin abans no es creï la conversa.

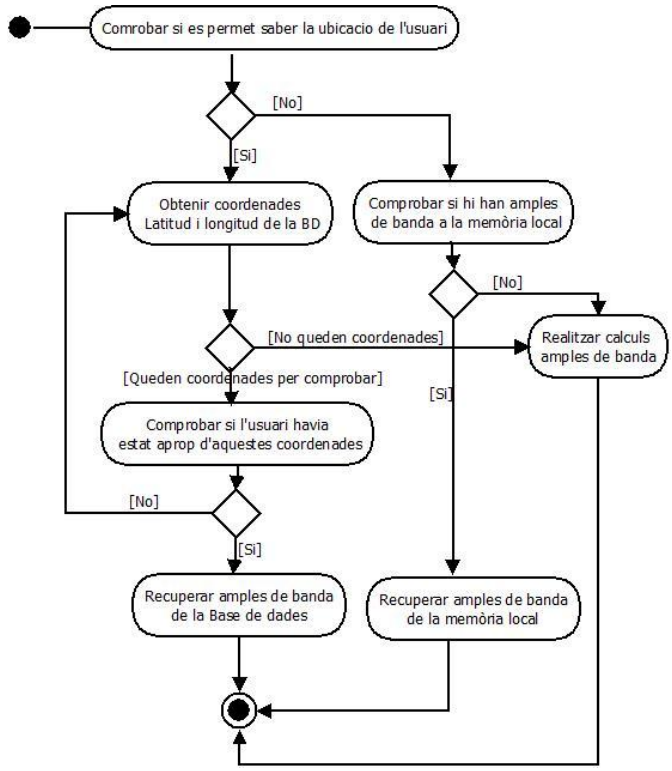
D'aquesta manera, en el moment en que tots els usuaris accepten una petició es crea la conversa.

Cas d'ús llistar peticions conversa:



Mostra les peticions enviades i rebudes de l'actor. Per simplificar la búsqueda l'actor pot entrar un nom i se li mostraran només les peticions en que hi hagin usuaris que continguin aquell nom.

Cas d'ús ubicar usuari:

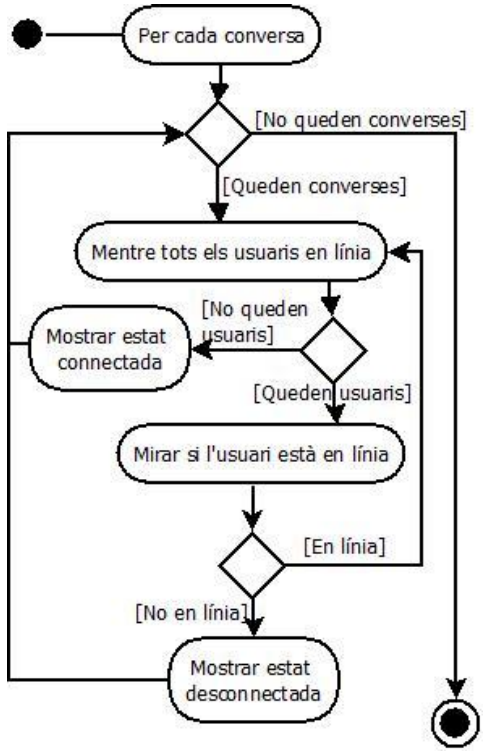


Aquest cas d'ús és llançat pel propi sistema al cap de 5 segons després de que l'usuari hagi entrat a la pàgina. La seva finalitat és obtenir la localització actual de l'usuari i les VDT que té la xarxa en la que es troba.

Per fer aquesta funcionalitat cal tenir en compte que l'usuari pot no voler donar la seva localització. En aquest cas s'usa la memòria del navegador per obtenir guardar els resultats del test de velocitat.

Si l'usuari dona la localització els resultats s'obtenen i es guarden a la BD.

Cas d'ús comprovar connectivitats:



Aquest cas d'ús el llança el sistema per a comprovar, per a una conversa determinada, si tots els usuaris que i participen estan en línia.

Com que l'usuari pot participar en varies converses quan entra a la pàgina realitza el test per totes les converses.

Llavors, cada cop que l'usuari selecciona una nova conversa es torna a realitzar el test.

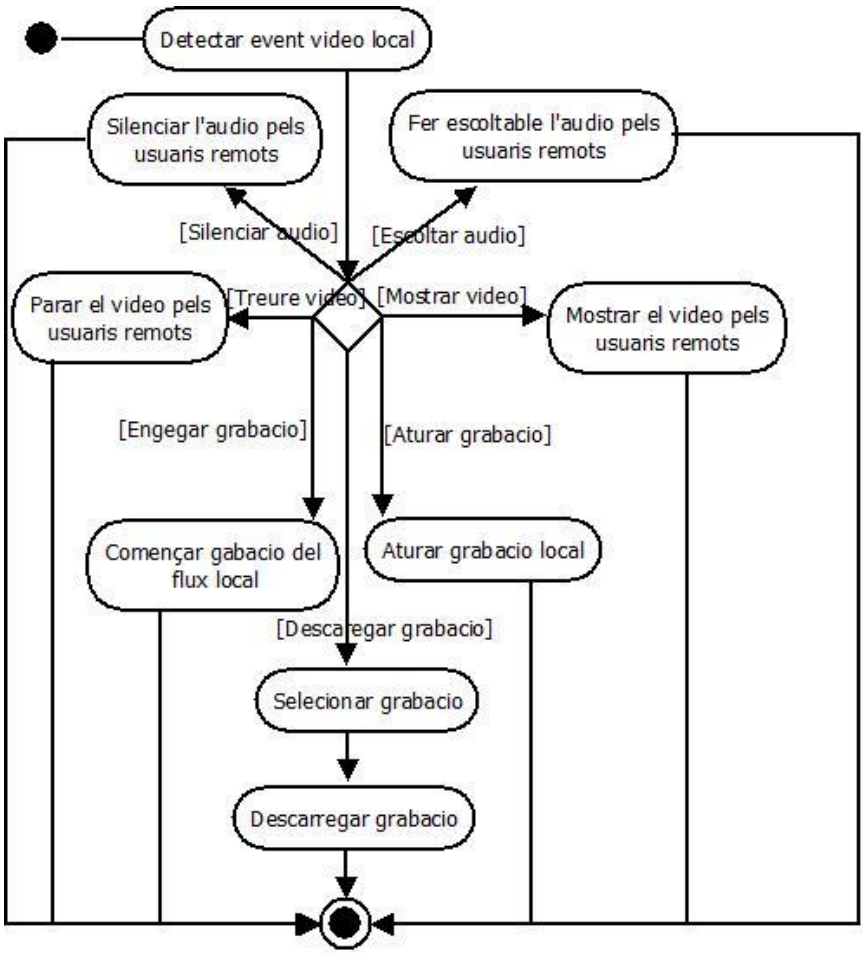
DIAGRAMES D'ACTIVITAT DE LA VIDEOCONFERÈNCIA

Seguidament,igual que en els diagrames anteriors primer es mostra una imatge per ubicar els esdeveniments que activen els casos d'ús:

Figura 17

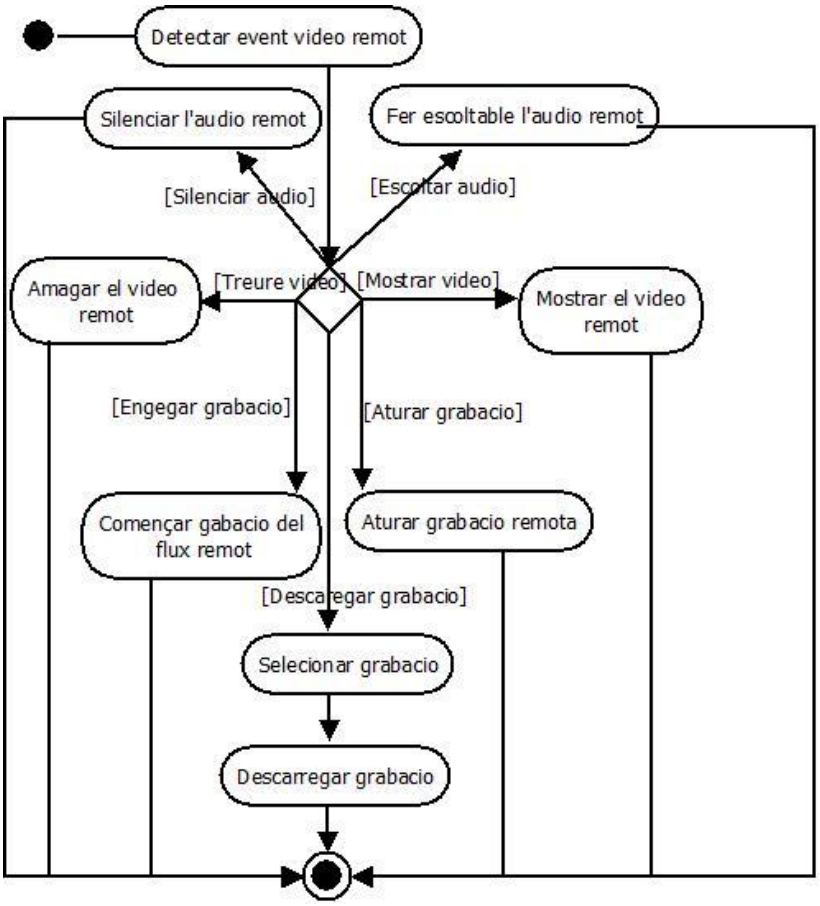
Els diagrames d'activitat són els següents:

Cas d'ús gestionar usuari local:

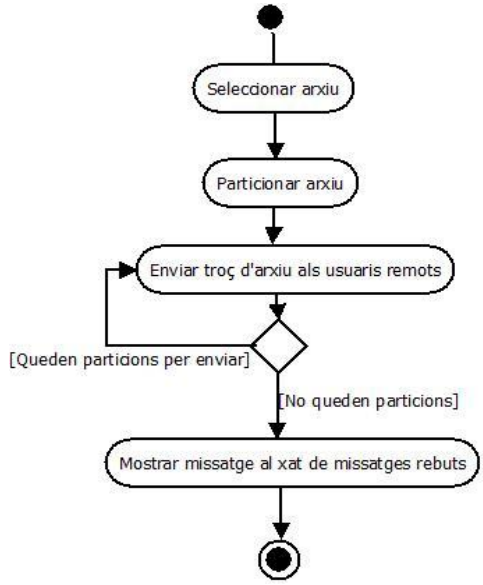


Aquest cas d'ús, al igual que per la gestió dels usuaris remots, podria separar-se en varis, però al tenir poques activitats per a cada funcionalitat s'ha optat per unir-ho tot en un.

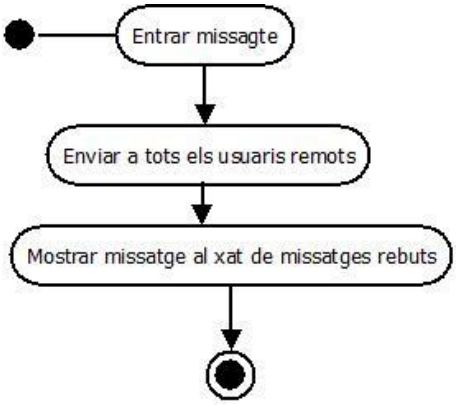
Cas d'ús gestionar usuari remot:



Cas d'ús enviar fitxer videoconferència:

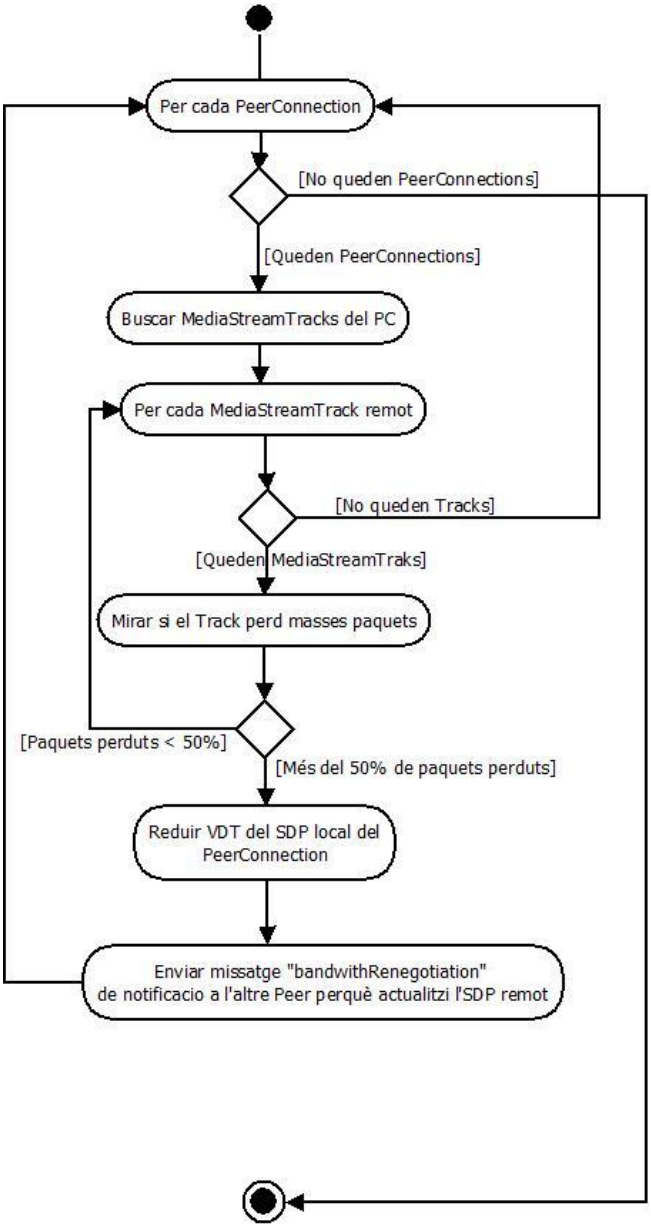


Cas d'ús enviar missatge videoconferència:



Aquestes dos últimes funcionalitats són molt semblants a les de l'anterior pàgina en que s'enviaven esdeveniments entre els usuaris d'una conversa. A diferència de les altres dos funcionalitats però, aquí els missatges s'envien a través del canal P2P de *WebRTC* i al no passar pel servidor no es guarden a la BD. Aquesta és la causa del perquè el xat de la videoconferència no és persistent.

Cas d'ús gestionar VDT:



Aquest cas d'ús l'activa el sistema cada deu segons per comprovar si en els últims deu segons s'ha perdut més d'un cinquanta per cent dels paquets en un *MediaStreamTrack* rebut. Si això passa vol dir que el *PeerConnection* és inestable i s'han de reduir les qualitats dels fluxos que rebem per la connexió *WebRTC*.

Per reduir la qualitat es disminueixen les VDT de l'objecte SDP local i s'envia el nou SDP a l'altre usuari perquè l'actualitzi, ja que sinó els canvis no tindrien efecte.

Aquests són tots els diagrames d'activitat pels possibles esdeveniments llançats per l'usuari. Les úniques funcionalitats que no s'han tingut en compte són les dels esdeveniments per a canviar de pàgina ja que en el codi és una simple redirecció a la URL indicada. Per tant, la fase d'anàlisi de requeriments es dona per acabada.

7. Estudis i decisions

Pel projecte no s'ha usat cap plataforma, degut a que el suport que donen els navegadors a WebRTC canvia constantment i si s'usa una plataforma per a videoconferències depens molt de que ells vagin actualitzant-la d'acord amb el suport.

A partir d'aquí s'ha construït una API (ASWRTC) pròpia (per la part client) per a l'ús de videoconferències directes basades en *WebRTC* i a sobre de la plataforma (API) que permet poder realitzar multi conferències entre els usuaris d'una conversa s'han construït els controladors i interfícies per a l'usuari.

L'idea de **construir aquesta llibreria sobre *WebRTC* és que tingués un molt baix acoblament amb la resta de l'aplicació** per tal de poder ser reutilitzada per qualsevol altre sistema que vulgui implementar videoconferències.

Ara es mostraran primer una introducció sobre les decisions preses a la part del servidor i llavors, passant a la part del client, es mostraran l'estudi i decisions per a la construcció de l'API de multi conferències, juntament amb les decisions preses per a la programació de la part *front-end* i *back-end* per a la interacció de la plana Web amb l'usuari.

7.1. Part servidora

En la part del servidor es necessitava una base de dades que permetés guardar usuaris, amistats, converses, missatges enviats entre els usuaris de les converses, configuracions de les videoconferències i els amples de banda màxims que pot utilitzar cada usuari per a les trucades multimèdia. Per fer això s'emmagatzemen les dades amb una base de dades relacional gestionada amb **MySQL**.

S'ha escollit una base de dades relacional pel fet de que el contingut de la base de dades tampoc és preveu que sigui molt gran i perquè no es té prevista cap fragmentació de la base de dades en varis servidors.

També cal un servidor **HTTPS** per proveir el contingut de les pàgines de la Web i permetre al client llençar peticions a la base de dades de manera segura. Per això s'ha instal·lat un servidor **Apache** per a les peticions HTTP, i **OpenSSL** per poder treballar en HTTPS.

Per poder passar paràmetres entre pàgines i fer comprovacions d'accessos s'ha instal·lat el llenguatge de programació de part del servidor **PHP**.

Per afegir totes aquestes funcionalitats a Ubuntu és necessari que en el procés d'instal·lació del SO, en el moment en que demana els paquets que es volen afegir per defecte, es marquin les opcions d'instal·lar els següents paquets (si no es fa això al principi llavors cal instal·lar manualment cada funcionalitat per separat):

- OpenSSH (que al instal·lar les dependències instal·la OpenSSL també)
- Una interfície per a la més senzilla gestió del servidor (per exemple Descktop gmond)
- Un servidor DNS per a no haver de referir-se el servidor a través de l'@IP
- Instal·lar un servidor LAMP; al instal·lar LAMP ja s'instal·la Apache, MySQL i PHP entre d'altres.

Finalment, per poder manejar la base de dades MySQL a través d'una interfície s'ha instal·lat *phpMyAdmin*.

Un cop instal·lada tota la part necessària per a la gestió de la base de dades i l'obtenció de pàgines del servidor cal afegir també un **servidor Node.JS** [21].

El servidor Node.JS ubicat al port 8080 s'utilitza per a poder establir un protocol Socket.IO entre client i servidor i per a proporcionar una API amb la qual el client pugui consultar, afegir, modificar i eliminar dades de la base de dades.

S'ha escollit l'entorn de programació *Node.JS* per la seva capacitat de fer peticions asíncrones a la base de dades i la facilitat en la que pots importar mòduls com el de Socket.IO per estendre les seves funcionalitats inicials.

Els mòduls usats en *Node* estan descrits anteriorment.

API en *NodeJS*

Gràcies als mòduls ***expess*** i ***MySql*** s'ha construït la interfície per a gestionar la base de dades. El mòdul *MySQL* permet assignar a una variable la opció de connectar-se amb la base de dades de la següent manera:

variable a la que se li ha assignat el mòdul *MySQL*

```
var client = mysql.createPool({
    host: 'localhost',
    user: 'usuariMySQL',
    password: 'contrasenyaMySQL',
 database: 'nomBaseDeDades'
});
```

Un cop obtinguda variable *client* per a gestionar la BD ja es poden llançar peticions a la base de dades:

obtenim connexió amb la BD

llancem una petició a la BD

```
client.getConnection(function(err, connection){
  connection.query(peticioMySQL,function(err, rows, fields) {
        //Codi (a rows hi ha el resultat de la petició)
        connection.release();  //alliberem la connexió amb la BD
   });
}
```

Per a crear un mètode de la API per a que la part client pugui interactuar amb la base de dades és on s'usa el mòdul ***express***:

```
require('express')().get('nomMèodeAPI', function(req,res){
  //si el mètode fos POST o PUT, els paràmetres estan dins de req.params
  //peticions i accions amb la BD
   res.json(resposta); //resposta al client
});
```

Socket.IO

Es va escollir Socket.IO perquè inicialment donés suport a la restricció del protocol de senyalització, però el seu ús s'ha estès i s'usa per a mantenir els usuaris informats de tots els esdeveniments referents a ells en temps real.

El conjunt de mètodes d'escolta (*socket.on*) que el servidor proporciona a la part client permeten enviar missatges entre usuaris sense que el client hagi de fer peticions HTTP al servidor constantment i amés, des dels mètodes (*socket.on del servidor*) es llancen tot tipus de "querys" a la base de dades, fent així que es pugui modificar la base de dades a través de l'API amb peticions HTTPS o a través del *socket* amb l'enviament de dades pel canal TCP bidireccional.

7.2. Part client

Per a la part client podem diferenciar entre la part de construcció de l'API de videoconferències i la part relacionada amb la interfície de l'usuari en que s'inicien i s'acaben tots els casos d'usos exposats en l'anàlisi de requeriments.

7.2.1.API de programació sobre *WebRTC* (ASWRTC)

La idea de construir un sistema de videoconferències en WebRTC es tenia des del principi, es va escollir la tecnologia de WebRTC pel fet de que permet realitzar videoconferències directament des del navegador sense la necessitat de l' utilització de "plug-ins", amés WebRTC, gràcies als ICE Candidats proporciona una arquitectura P2P que envia els fluxos multimèdia directament entre els usuaris que participen a la videoconferència.

Les funcionalitats que proporciona l'API **ASWRTC** són la gravació de fluxos, el control d'enviament i rebuda de fluxos multimèdia, la regulació de les velocitats de transmissió de dades per a que no es sobresaturi la xarxa, enviament de missatges a través d'un *PeerConnection* i l'obtenció d'estadístiques referents a cada canal P2P entre dos usuaris.

Ara es mostren les llibreries usades en cada funcionalitat que ASWRTC proporciona:

CONTROL DELS FLUXOS MULTIMEDIA

Per cada usuari que intervé a la conversa s'estableix una connexió P2P entre ell i tots els demés fent així que quedi, en una multi conferència de quatre usuaris un esquema com el següent:

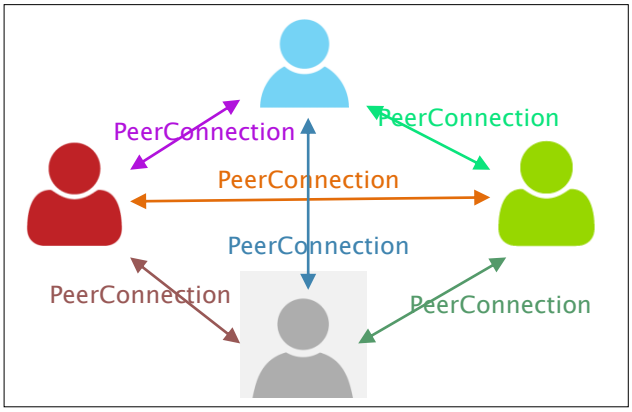


Figura 18

Per el control de fluxos multimèdia no s'utilitza cap més llibreria que l'API de *WebRTC*. Per a cada flux que es rep d'un PeerConnection es té una opció de control (parar/engegar àudio i parar/engegar vídeo), i per controlar els fluxos multimèdia que s'envien es té una única opció de control que actua sobre tots els fluxos de tots els *PeerConnections*.

Si un usuari entra sense vídeo o àudio se li dona l'opció també d'engegar-lo durant la conversa, aquest fet és una mica més complexa que els anteriors ja que al iniciar per primer cop un MediaStreamTrack que no havia estat creat prèviament cal refer totes les connexions WebRTC. Aquest és l'únic cas en que és necessari esborrar i crear de nou tots els objectes *PeerConnections* que comuniquen l'usuari amb els demés usuaris de la videoconferència.

GRAVACIÓ DE DADES

Un usuari pot gravar l'àudio i/o vídeo local, així com l'àudio i/o vídeo remots provinents de qualsevol *PeerConnection*. Per a la gravació s'ha usat la llibreria **RecordRTC** [22] [23] que proporciona la possibilitat de gravar tots els fluxos multimèdia de qualsevol objecte **MediStream** ja sigui local o remot.

En l'enllaç *22* és mostra el codi de la llibreria i se n'explica el funcionament i en l'enllaç *23* hi ha una "demo" de funcionament.

La llibreria dona el següents suports:

Browser	Support	Features
Firefox	Stable / Aurora / Nightly	Audio+Video (Both local/remote)
Google Chrome	Stable / Canary / Beta / Dev	Audio+Video (Both local/remote)
Opera	Stable / NEXT	Audio/Video Separately
Android	Chrome / Firefox / Opera	Audio/Video Separately
Microsoft Edge	Normal Build	Only Audio

En *Chrome* i *Firefox* la llibreria funciona correctament en tots els casos, excepte que en *Chrome* no es permet la gravació de vídeo sol.

Per a gravar utilitzant la llibreria, en el nostre cas, tant en *Chrome* com en *Firefox* li passem al mètode *RecordRTC* l'objecte *MediaStream* del que volem gravar els seus Tracks (àudio i vídeo) i una llista d'opcions. Aquestes opcions tenen en compte si el *MediaStream* té àudio i/o vídeo.

Les opcions que li passem al mètode *RecordRTC* són les següents depenent del cas:

- L'objecte *MediaSteam* té només àudio:
    - type: 'audio' (indiquem que només gravem àudio)
    - audioBitsPerSecond: 128000 (numero de bits per segon que gravem)
- L'objecte *MediaSteam* té només vídeo:
    - mimeType: 'video/webm' (indiquem el format del vídeo de sortida)
    - videoBitsPerSecond: 128000

- L'objecte *MediaSteam* té àudio i vídeo:
    - mimeType: 'video/mp4' (indiquem el format del vídeo i àudio de sortida)
    - bitsPerSecond: 128000

OBTENCIÓ D'ESTADÍSTIQUES

L'obtenció d'estadístiques sobre cada *PeerConnection* permet donar una idea de com està funcionant una videoconferència sense la necessitat de ser-hi present. Les estadístiques proporcionades per cada PeerConnection són els ICE candidats escollits, el numero de paquets perdut, les velocitats de transmissió que s'han usat i la diferència entre les velocitats de transmissió que s'han usat durant la multi conferència i els límits imposats.

Per a l'obtenció d'estadístiques s'ha usat la llibreria **GetStats** [24] i s'ha readaptat per a que funciones correctament en les últimes versions de ***Chrome*** **(52.0.2743.116)** i ***Firefox*** **(48.0)**.

La llibreria getStats utilitzada permet obtenir tots els paràmetres que es necessiten tot i que ho fa d'una manera diferent segons Chrome o Firefox.

En el cas de Chrome podem obtenir l'estat actual amb un sola crida:

```
RTCPeerConnection.prototype.__getStats = window.getStats;
ObjectePeerConnection.__getStats(ObjectePeerConnection,function(result) {
    //Resultat conte les estadístiques
}
```

La llibreria proporciona un altre mètode en que et dona totes les estadístiques en paràmetres detallats però no funciona en les últimes versions de Chrome i Firefox. Per això es crida aquest mètode intern de la llibreria. Al cridar el mètode intern les estadístiques que es proporcionen són les acumulades durant tota la sessió, a partir d'aquí es passen les dades respecte cada segon.

Cada 10 segons es crida aquest mètode i s'actualitzen les estadístiques mitjanes del *PeerConnection* per a la videoconferència.

En el cas de Firefox s'ha d'obtenir l'estat per a cada MediaStreamTrack apart:

```
RTCPeerConnection.prototype.__getStats = window.getStats;
ObjectePeerConnection.__getStats(MediaStreamTrack,function(result) {
    //Resultat conte les estadístiques
}
```

En aquest cas alguns dels paràmetres que s'obtenen són diferents, així com hi han paràmetres que teníem en *Chrome* i *Firefox* no, però els dos paràmetres més importats pels ICE Candidates (bytesSend i bytesReceived) estan per igual a *Chrome* i *Firefox*.

Independentment del navegador l'argument *"result"* és un objecte que conté entre d'altres la llista de ICE Candidats (tan els usats com els que no), a partir d'aquí es busquen els candidats usats per a cada *track* i a dintre de l'objecte si poden trobar els camps ***bytesSent*** (per un

MediaStreamTrack local, conté el total d'informació enviada pel MediaStreamTrack en bytes), ***bytesReceived*** (per un MediaStreamTrack remot, conté el total d'informació enviada pel MediaStreamTrack en bytes), **local** (diu si el candidat és local o remot)...

Els paràmetres que s'han utilitzat de la llibreria són els següents:

- **local**
  Indica si el *MediaStreamTrack* és local (l'enviem nosaltres) o és remot (el rebem).
- **bytesSent**
  Per un *MediaStreamTrack* enviat obtenim el numero de bytes enviats durant tota la sessió.
- **bytesReceived**
  Per un *MediaStreamTrack* que rebem obtenim el numero de bytes rebuts durant tota la sessió per aquest flux d'àudio o vídeo.
- **packetsSent**
  Per un *MediaStreamTrack* que enviem obtenim el numero de paquets enviats durant el transcurs de tota la sessió.
- **packetsReceived**
  Per un *MediaStreamTrack* que rebem obtenim el numero de paquets rebuts durant el transcurs de tota la sessió.
- **packetsLost**
  Indica per un *MediaStreamTrack* (local o remot), el numero de paquets perduts durant tota la sessió. Aquesta variable és calcula a partir de la resta entre els paquets que s'estima que s'han d'enviar o rebre pel *MediaStreamTrack* i els paquets que realment s'envien o es reben (*packetsSent* o *packetsReceived*).
- **googBandwithLimitedResolution**
  Indica en un moment donat si s'està limitant la resolució d'un *MediaStreamTrack* de vídeo. Aquesta variable és útil per saber si al limitar la VDT de l'objecte estem limitant realment la resolució del vídeo o realment no estem fent res.

Aquests paràmetres descriuen les estadístiques de cada objecte *MediaStreamTrack* i per una PeerConnection podem obtenir en el nostre cas fins a quatre *MediaStreamTrack* diferents (un per l'àudio enviat, un pel vídeo enviat, un per l'àudio rebut i un pel vídeo rebut).

La API ASWRTC proporciona al programador que l'usa tots aquets paràmetres especificats però també els utilitza per ella mateixa. **La API utilitza els paràmetres *packetsLost* i *packetsReceived* per determinar si el flux rebut a través d'un *MediaStreamTrack* és estable o no s'està rebent fluidament**. Si després d'un interval de deu segons el numero de paquets perduts (*packetsLost*) és major al numero de paquets rebuts (*packetsReceived)* significa que s'està perdent més d'un 50% dels paquets i que per tant en els últims 10 segons s'han rebut les dades del *track* molt tallades. A partir d'això la API sap que l'usuari que detecta el problema li ha demanat a l'altre usuari del *PeerConnection* que li envies una quantitat de dades que (o bé ell no pot enviar, o bé aquest usuari no pot descarregar).

La solució a aquest problema està en obligar a l'altre usuari a disminuir la quantitat de bits per segon que envia (per fer això és quan cal modificar les velocitats de transmissió l'objecte SDP local en viu i notificar-li a l'altre usuari el canvi).

Aquest canvi pot implicar tant la disminució de la resolució de vídeo que rebem com la qualitat de l'àudio que escoltem de l'altre usuari, però és l'única solució al problema de connexions inestables.

REGULAR LES VELOCITATS DE TRANSMISSIÓ

Un usuari que estableix una connexió WebRTC amb un altre pot regular les velocitats de transmissió dels fluxos multimèdia tant de pujada com de baixada.

Inicialment quan és va voler oferir connexions WebRTC entre tots els usuaris d'una multi conferència ja se sabia que hi hauria un fort problema d'escalabilitat en els navegadors dels clients. Per solucionar-lo es va pensar en reduir la qualitat dels vídeos a mesura es feien videoconferències més nombroses per tal de que al reduir la qualitat, no fos necessari processar tants bits per segon i s'hagués d'enviar menys informació.

Una altre opció era modificar el *frameRate* del vídeo i àudio, però per sota de 20 *fps* es començava a veure el processament d'imatges i la veu se sentia tallada, per **això s'ha establert fixament el numero d'imatges per segon a 24 fps.**

Un cop fixat el numero d'imatges per segon que es processen es va optar per una opció molt semblant a la primera per tal de limitar la quantitat d'informació enviada. Es va aprofitar que en WebRTC es pot limitar la velocitat de transmissió a nivell de xarxa per tal d'ajustar la qualitat del vídeo enviat a la capacitat que li permet la xarxa a l'usuari. D'aquesta manera els usuaris ubicats en xarxes amb més velocitats de transmissió podran realitzar intercanvis de fluxos multimèdia de més qualitat.

Per exemple si partim d'una càmera Full HD que representa les imatges en un detall de 1920×1080 píxels necessitem una taxa de bits aproximada d'enviament de 5000Kbps (depèn del còdec de vídeo que s'utilitzi el *frameRate* del vídeo). Això significa que la nostre xarxa, suposant que volem rebre també senyal Full HD, a de tenir disponible una velocitat de transmissió de pujada de 5000Kbps i una de baixada de 5000Kbps només per vídeo (sense comptar àudio). Les xarxes asimètriques dels domicilis particulars solen tenir una VDT (comptant un cas bastant dolent) de 788.48kbps de pujada i 10240kbps de baixada aproximadament. Per tant des d'un domicili particular, si es fes una videoconferència amb només un altre usuari i els dos tinguéssim càmeres Full HD, l'usuari particular podrà rebre el vídeo remot en Full HD sense interferències però només podrà enviar un flux fluït cap a l'altre amb com a molt una taxa de 788.48kbps i per tant, encara que la seva càmera sigui Full HD, s'ha de limitar l'enviament de vídeo a una resolució molt menor a la que té.

Abans d'enviar els fluxos multimèdia el senyal passa pels còdecs de comprensió que els comprimeixen. Així doncs **la qualitat del vídeo que s'envia va molt relacionada amb el còdec usat**. Per saber la resolució que proporciona un còdec de vídeo per un bitrate determinat es poden utilitzar les variables *frameWith* i *frameHeight* de la llibreria ***getStats*** de WebRTC que indiquen la resolució a la que s'envia o es rep un flux de vídeo.

En el cas de *WebRTC*, en el intercanvi dels SDP s'especifica una llista de prioritat dels còdecs que s'usaran per a cada medi. Aquesta llista es pot modificar, però per defecte la prioritat dels còdecs és la següent:

Medi d'àudio:

ISAC
G722
PCMU
PCMA
CN

Medi de vídeo:

VP8
ccm fir
VP9
H.264

Pel que fa el vídeo tant VP8 com H.264 necessiten aproximadament 0.1427 bits per a cada píxel d'informació, a això se li ha de tenir en compte el numero de cada frame del vídeo i un *frameRate* de 24 imatges per segon per a saber el nombre de bits per segon que necessita.

A partir d'això es va optar per a donar l'**opció de que al cridar la llibreria se li poguessin passar dos parells d'amples de band**a tenint en compte els amples de banda que proporcionen els còdecs, **el primer parell indica la velocitat de transmissió d'àudio i vídeo de pujada i el segon és per la velocitat de baixada pels dos medis (àudio i vídeo).**

A partir d'aquí es modifica els SDP local tal com sindica prèviament en l'apartat del protocol SDP per a cada *PeerConnection*. Per a modificar l'SDP s'usa la següent llibreria ***BandwithHandler*** [25]. La llibreria es crida un cop s'ha cridat el mètode ***createOffer*** o ***createAnswer*** i s'ha obtingut la descripció SDP. A aquesta descripció, abans d'establir-la com a definitiva i enviar-la a l'altre peer se li aplica els següents mètodes de la llibreria:

```
var bandwidth = {
        screen: "velocitat de transmissió màxima per una compartició de pantalla"
        audio: "velocitat de transmissió màxima per l'àudio"
        video:  "velocitat de transmissió màxima pel vídeo"
};
var isScreenSharing = false; //indiquem que no compartim pantalla
description.sdp=BandwidthHandler.setApplicationSpecificBandwidth(description.sdp,
                    bandwidth, isScreenSharing);
description.sdp = BandwidthHandler.setVideoBitrates(description.sdp, {
    min: bandwidth.video,
    max: bandwidth.video
});
description.sdp = BandwidthHandler.setOpusAttributes(description.sdp);
```

Aquesta modificació es fa sobre els SDP locals i indica la velocitats de transmissió d'enviament i rebuda dels medis d'àudio i vídeo remots. Per tant, per l'usuari remot, les VDT de l'SDP local pels medis d'àudio i vídeo seran les seves VDT de pujada.

En la secció 15.3 (annexes) hi ha una explicació sobre els fluxos multimèdia que s'envien i la velocitat de transmissió (ample de banda) [que fan quan ens limiten l'ample de banda].

ASWRTC també té la capacitat d'autoregular les VDT si detecta que els fluxos que rep un *PeerConnection* no són estables. Per detectar el problema s'utilitza l'obtenció d'estadístiques tal com s'ha explicat en la secció anterior, però un cop detectat que s'estan perdent masses paquets d'un MediaStreamTrack de rebuda, llavors ASWRTC modifica les VDT del SDP local i envia l'objecte SDP local perquè l'usuari remot sigui conscient de l'actualització i actualitzi l'objecte SDP remot.

Per fer aquesta actualització de la descripció de sessió local en temps real s'obté i s'utilitza l'SDP local que està a la variable *localDescription* de l'objecte *PeerConnection*, i de nou amb la llibreria ***BandwidthHandler*** es modifiquen les VDT antigues per les noves.

El que fa la llibreria és esborrar totes les limitacions que hi havien anteriorment i establir les noves limitacions per cada medi d'àudio i vídeo.

Tot i que es pot limitar cada MediaStreamTrack per separat, quan es detecta que hi han problemes d'estabilitat per un *MediaStreamTrack* rebut, automàticament es baixen les VDT de tots els *MediaStreamTrack* rebuts del *PeerConnection*. Això es fa així perquè quan hi ha problemes d'aquests tipus les xarxes dels dos usuaris solen estar molt saturades i s'intenta acabar amb el problema el més dràsticament possible. **El sistema va baixant les VDT cada cop que el *PeerConnection* és sobresatura fins a un mínim de 8kbps pel medi d'àudio i 16kbps pel de vídeo**.

CONTROL DELS DISPOSITIUS DE L'USUARI

L'API ASWRTC té un control dels dispositius dels que disposa l'usuari d'una videoconferència, així com si permet utilitzar-los o no). La idea d'això és poder informar a als usuaris en casos com els que hagin demanat retransmetre senyal de vídeo i no tinguin cap càmera disponible. Per això s'utilitza la llibreria **DetectRTC** [26] [27].

El funcionament de la llibreria està explicat en l'enllaç [26] i en l'enllaç [27] hi ha una demo que en demostra el funcionament.

7.2.2.Interacció entre l'aplicació i l'entorn

Per a construir la interfície Web s'ha usat HTML i CSS com a *front-end* i JavaScript com a *back-end*.

Per la part de Javascript s'han usat la llibreria ***JQuery*** [28] i el Frameworks ***AngularJS*** [29] que permeten interactuar d'una manera més senzilla amb el *front-end*. Al principi no es tenia previst utilitzar AngularJS però veient la complexitat que donava comunicar les parts *front-end* i *back-end* es va decidir passar a fer-ho pràcticament tot en AngularJS.

JQuery

L'ús de JQuery és pel fet de simplificar la interacció amb el document HTML i s'usa especialment per a la modificació del document CSS en temps d'execució gràcies al mètode $("identificadorCSS").css("nomEtiqueta","estil"). A part d'això també s'usa la llibreria ***JQuery-min***

per a introduir estils HTML ja predefinits, com per exemple, per a les barres de carregant/enviant fitxers, s'usen funcionalitats d'aquesta llibreria.

AngularJS

L'entorn de programació Angular s'usa per a la programació de les classes controladores de l'aplicació i per poder comunicar la vista amb l'API REST, de manera que les dades mostrades estiguin sempre actualitzades.

Angular és un Framework MVVM en què es pot distingir un Model, una Vista i un VM. VM és un objecte vista-model que estén les funcionalitats d'un controlador típic creant un pont que comunica la vista amb el model, per el qual vista i controlador queden comunicats de tal manera que quan hi han canvis a la vista s'actualitza immediatament el controlador i viceversa.

L'ús d'angular és degut a que l'aplicació presentava les següents característiques:

- Les vistes es constitueixen a partir de dades del model.
- Les vistes modifiquen dades de l'aplicació i viceversa.
- L'aplicació consumeix dades d'una API Rest i volem comunicar el *front-end* amb el *back-end* de manera directe.
- La lògica de JavaScript és bastant complexa

Els quatre conceptes sobre els que gira *AngularJS* i que s'han usat a SRTM són:

- Directives (permet afegir funcionalitats a l'arxiu HTML, de manera que per exemple podem afegir una directiva que permeti detectar l'esdeveniment de quan l'usuari prem la tecla enter).

- Controladors (contenen la lògica de l'aplicació, permeten comunicar de manera senzilla la part *front-end* i *back-end* gràcies en part a l'objecte ***$scope***, que és la representació de la vista dins del controlador, gràcies a aquest objecte podem modificar la vista des de JavaScript i viceversa).

- Factories i Serveis (permeten crear codi compartit entre varis controladors d'un model, en una Factory es crea un objecte personalitzat que al igual que l'objecte *$scope* pot ser accedit des de qualsevol controlador).

- Filtres (permeten fer vistoses dades en formats que no estan preparats per a l'usuari, com per exemple els objectes *Date* de *JavaScipt* i *MySQL*).

Dit això, AngularJS és un entorn de programació modularitzat que permet tant importar funcionalitats d'altres mòduls com la creació de mòduls personalitzats. En el nostre cas, a cada pàgina Web tenim un mòdul diferent referenciat al inici del document HTML (amb la comanda ng-app="nommodul") i per cada controlador, factoria o directiva que pertany al mòdul.

L'únic mòdul extern que s'utilitza és el ***LocalStorageModule*** [31] que permet guardar i recuperar dades en la memòria local del navegador perquè persisteixin entre sessions.

Amés dels mòduls, a cada pàgina hi ha un o més Controladors associats al mòdul, el Controlador s'associa a les vistes posant ng-controller="nomControlador" en el tag dintre del qual volem tindre una vista i un controlador.

Per a cada controlador podem importar Serveis i Factories que proporciona el *FrameWork* d'Angular per defecte. En el nostre cas s'usa ***$filter*** (permet aplicar funcions per filtrar els

elements d'un vector que compleixin unes determinades característiques) i ***$http*** (s'usa per a fer peticions asíncrones AJAX en Javascript a un servidor HTTP, gràcies a aquest servei podem fer peticions HTTPS per obtenir dades permanents guardades a la BD).

Per l'ús d'Angular i JQuery cal importar les llibreries de JavaScrit *jquery.min*, *jquery-ui.min*, *angular-local-storage.min*, *angular.min*.

ÚS DE L'API SOBRE WebRTC (ASWRTC)

En la parlat anterior s'ha parlat de la construcció de la API sobre WebRTC (ASWRTC) que proporcionava les funcionalitats necessàries per a la creació i control d'un sistema basat en videoconferències *WebRTC*.

La llibreria obliga al programador (entre d'altres) **a passar-li les velocitats de transmissió inicials** que vol usar per a pujar i baixar els fluxos dels diferents medis. Llavors **si aquestes VDT són excessives la API les disminuirà** (però si no es passen unes VDT correctes, en el període inicia, fins que l'API no detecti el problema i el solucioni, hi hauran problemes d'estabilitat).

Així doncs es necessari un sistema per a obtenir els amples de banda que s'usaran en cada videoconferència per passar-los-hi a la API ASWRTC.

Per seleccionar les velocitats de transmissió de pujada i baixada adients per a cada usuari es varen tenir en compte els següents dos aspectes:

- L'usuari no hi ha d'intervenir (ha de ser automàtic)
- Ha de ser adaptable a la xarxa que usa l'usuari

A partir d'aquí es presentava el problema de com saber els amples de banda que té la xarxa d'un usuari.

Actualment aquest problema no té una solució directe ja que no es permet des de cap llenguatge de programació poder obtenir dades sobre la xarxa en la que està un usuari. Degut això es va optar per resoldre el problema de la següent manera:

- Fer un test de la velocitat de transmissió de pujada i baixada de l'usuari en segon pla.
- Determinar la xarxa en la que està l'usuari a partir de las seva geocalització.

Pel test de velocitat el que es fa és descarregar i pujar un arxiu de 10MB contant el temps que triga en cada acció, a partir del temps en que tarda a baixar l'arxiu es determina la velocitat de transmissió (en *kbits/s*) de baixada que té la xarxa de l'usuari, altrament, a partir del temps en que el client tardi a pujar l'arxiu es determina la velocitat de transmissió de pujada de la xarxa.

Aquest procediment proporciona les velocitats de transmissió de manera molt exacte, tot i que la complexitat temporal és més elevada ja que per exemple, sobretot alhora de pujar l'arxiu, per usuaris que tenen xarxes asimètriques amb capacitats de pujada menors als 32KB/s el test tardarà més de cinc minuts en acabar.

Per mitigar aquest problema de la complexitat temporal del test era necessari no haver de fer aquest test cada cop que l'usuari entra al sistema, per això els resultats del test es guarden persistentment tant a la base de dades, com en la memòria local del navegador.

Tot i això seguia havent-hi un problema; la funcionalitat no contemplava que un usuari estigues utilitzant un ordinador portàtil i podés autentificar-se des de xarxes diferents. Per solucionar això els amples de banda calculats es relacionen en amb les coordenades (Latitud i Longitud) en que es troba l'usuari, suposant que un cop s'ha fet el test, si l'usuari torna a entrar a l'aplicació des d'un punt proper a les coordenades des de les quals havia accedit abans estarà ubicat a la mateixa xarxa.

Per obtenir les coordenades en la que està un usuari s'usa l'API ***Geolocation*** [32] d'HTML5, l'API dona suport a tots els navegadors en la última versió i permet obtenir les coordenades, en forma de latitud i longitud, en que es troba el client, a partir del següent mètode:

```
navigator.geolocation.getCurrentPosition(function(position){
   //position.coords.latitude conté la latitud
   //position.coords.longitude conté la longitud
});
```

A partir d'aquí, en Chrome és necessari fer córrer un servidor pyton (`python -m SimpleHTTPServer`) en el mateix directori a on es troba l'arxiu JavaScript que demana la posició.

El problema que aquest sistema presenta es que si un usuari des d'un punt determinat es connecta a xarxes diferents llavors els amples de banda que se li assignen a la seva xarxa seran erronis, aquí apareix però, en el cas de que les VDT siguin excessives, la responsabilitat de modificar en temps real les VDT errònies per part de la API ASWRTC. Per altre banda, si les VDT escollides són molt menors al *bitrate* que pot utilitzar l'usuari la API ASWRTC no farà res per augmentar-les i ha de ser el programador que usa la llibreria qui, a través de l'obtenció d'estadístiques, detecti que les VDT que està escollint es podrien ampliar.

Però un cop sabem els amples de banda de pujada i baixada, com s'assignen les velocitats de transmissió que s'usaran en una videoconferència per als diferents medis i numero d'usuaris?

A partir de les velocitats de transmissió de pujada i baixada obtingudes en el test de velocitat el que es fa és (un cop l'usuari vol entrar en una videoconferència) primer agafar el **70% de tota la capacitat de pujada i el 60% per la baixada**, per deixar així la possibilitat de que l'usuari pugui executar altres aplicacions.

A partir d'aquí es divideixen les dos velocitats pel nombre d'usuaris que tindrà la videoconferència, ja que les velocitats de transmissió que se li passen a l'API s'apliquen per cada PeerConnection no per la suma de tots.

Un cop es té la velocitat de pujada i baixada per el intercanvi de flux amb un únic usuari s'ha de repartir pels diferents medis que intervindran en el flux. En aquest cas el que es fa és buscar en una **llista predefinida** que s'ha fet de VDT's el parell de velocitats de transmissió (pel medi d'àudio i vídeo) de pujada i baixada que no sobrepassen les velocitats de transmissió màximes.

La **llista predefinida** de velocitats de transmissió s'ha determinat a partir dels còdecs i aquest enllaç [30]. En la llista cada element és una tupla de dos valors, el primer valor de la tupla indica la velocitat de transmissió permesa per l'àudio i el segon pel vídeo. La llista té trenta

possibles velocitats de transmissió que van des de (8,16) kb/s per àudio i vídeo fins a (256,15360) kb/s.

Aproximadament, si s'usen els còdec VP8 i ISAC per codificar els medis d'àudio i vídeo respectivament, amb un *bitrate* de 15360 per la senyal de vídeo, tenint en compte que de la càmera s'obtenen 24 fotogrames per segon, es poden enviar imatges amb resolucions superiors a Full HD (1920x1080). Per l'àudio, si es codifiquen 256 bits cada segon amb el còdec ISAC la qualitat de l'àudio que s'envia també és molt elevada.

En el cas de que s'hagin d'usar bitrates de 8 i 16 kb/s, la poca qualitat que es proporciona en la veu no s'aprecia molt en una videoconferència en que els usuaris només parlen, però pel que fa el vídeo la resolució que s'utilitza és igual o menor a 320x240, cosa que fa que es vegi la imatge molt borrosa.

En el cas de que la xarxa de l'usuari no tingui ni 24kbs disponibles de pujada per cada usuari de la conversa no se li permet a l'usuari realitzar la videoconferència i se li ofereix la opció d'entrar només amb àudio per tal de que necessiti menys recursos.

8. Anàlisi i disseny del sistema

Fet l'anàlisi de requeriments aquest apartat va destinat a les fases d'anàlisi i disseny, deixant per l'apartat 10 la part referent a la implementació.

Per a les fases d'anàlisi i disseny s'usarà també, al igual que per la definició dels casos d'ús i diagrama d'activitats, el llenguatge UML per definir, a través de diagrames, el model final del projecte de cares a la part *front-end*.

Pel que fa la part *back-end* és mostrarà les parts d'anàlisi i disseny de la Base de dades juntament amb la API REST per a que el client pugui interactuar-hi.

8.1. Anàlisi del sistema

En aquest apartat es fa l'anàlisi de les classes necessàries que necessitarà la part client, i per la part del servidor es fa un anàlisi del que s'haurà de guardar a la base de dades i de les dades que hauran de ser proporcionades en temps real usant SOCKET.IO.

Per tal de situar com interactuen client i servidor a gran escala, primer es mosta l'arquitectura de SRTM.

8.1.1. Arquitectura

El sistema consta d'una part client i un servidor; els dos es comuniquen a través de peticions HTTPS i canals bidireccionals en Socket.IO. Per a la realització de converses WebRTC entre xarxes privades també es proporciona un servidor STURN/TURN.

L'arquitectura queda de la següent manera:

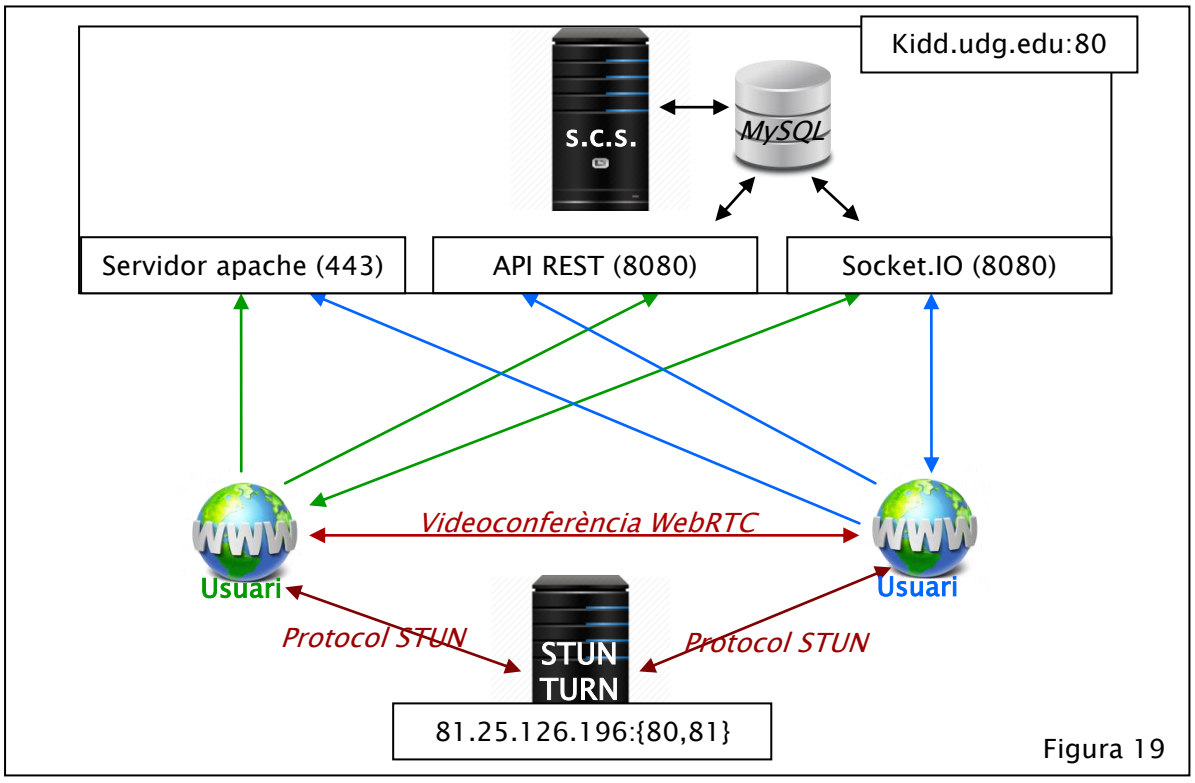


Figura 19

El client pot interactuar amb els ports 80, 443 i 8080 del servidor SCS.

El port 80 (HTTP no segur) només es pot usar per demanar la pàgina d'inici, llavors el servidor SCS redirigeix la petició al port 443 i carrega la pàgina amb HTTPS. El port 443 s'usa per a servir les pàgines del sistema a través del protocol HTTPS.

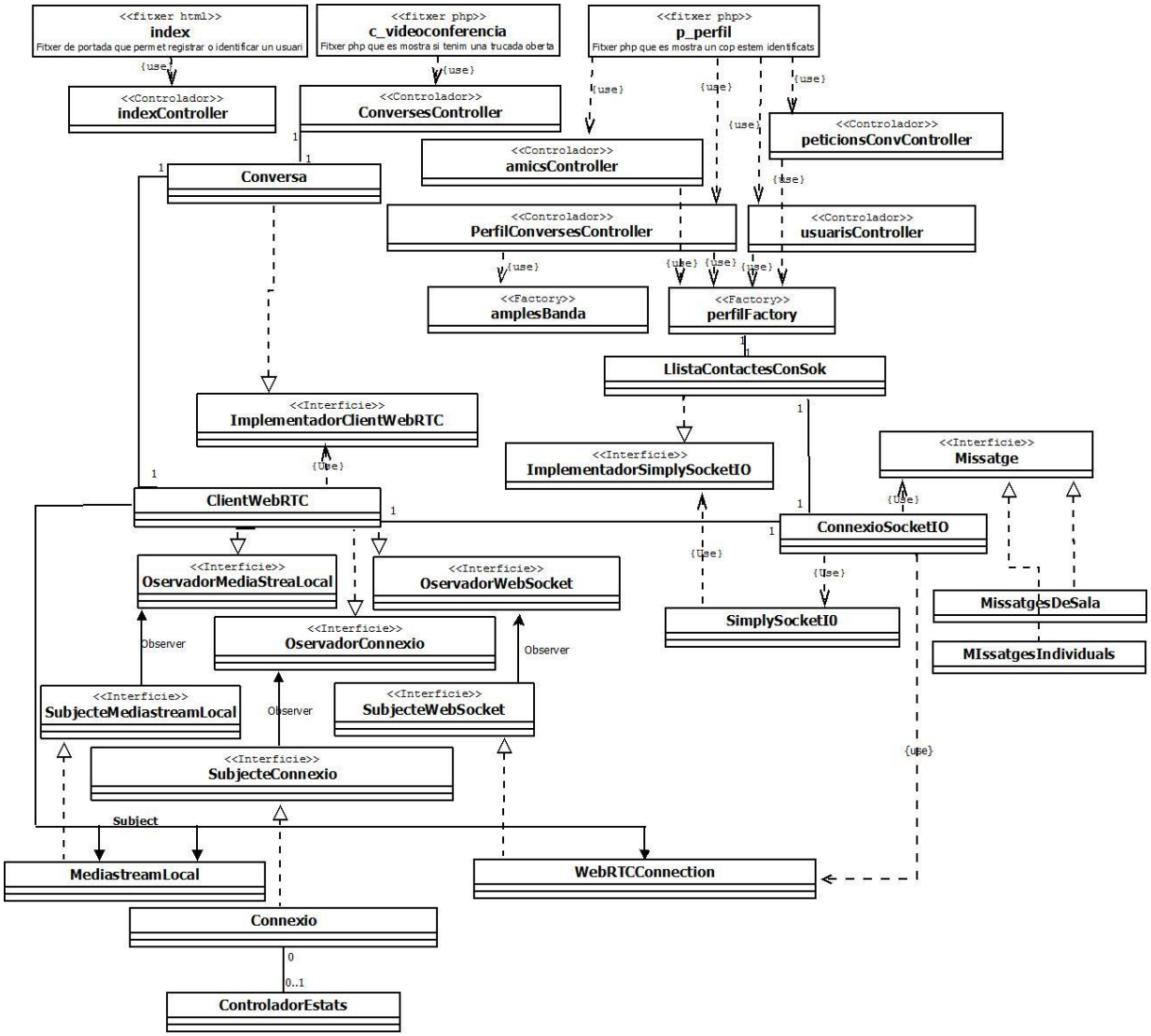
Per obtenir i modificar dades de la BD el client pot fer peticions HTTPS a l'API REST ubicada al port 8080 i a l'API de Socket.IO que es troba al mateix port. En cas d'usar l'API de Socket.IO per modificar la base de dades els clients afectats són informats dels canvis en temps real.

Quan dos o més usuaris estableixen una videoconferència aquesta es fa directe entre navegadors i només s'usa el servidor SCS per a fer la senyalització WebRTC que permet establir els canals P2P i per modificar les VDT en temps real.

Pel cas de que algun dels usuaris estigui en una xarxa privada alhora d'obrir un canal P2P per a la videoconferència amb un altre navegador és necessari usar el protocol STUN per a superar els routers NAT, o en el pitjor dels casos, redirigir la connexió WebRTC a través del servidor TURN.

8.1.2. Diagrama de classes fase anàlisi

El diagrama de classes per a la fase d'anàlisi és el següent:



Dintre del diagrama de classes es poden diferenciar tres blocs, el primer es per la part *front-end*, que compren totes les classes que estan per sobre de la classe *LlistaContactesConSock*. Per la part *back-end* tenim a la part esquerra el conjunt de classes que proporcionen els mètodes per a la creació de videoconferències (totes les classes que giren entorn de *ClientWebRTC* i la seva interfície *ImplementadorClientWebRTC*), aquest conjunt de classes constitueixen l'API ASWRTC, i a la part dreta el conjunt de classes que es comuniquen amb el servidor a través de Socket.IO (que composen l'API ASSIO) .

Per la part de videoconferència és necessari el protocol de senyalització per establir un canal WebRTC entre dos usuaris, per això la classe *WebRTCConnection* està relacionada amb el bloc per a la gestió de videoconferències, ja que s'encarrega de proporcionar els missatges Socket.IO per a la gestió de trucades.

Seguidament s'explica breument la funcionalitat de cada classe:

- índex

  És la interfície (HTML+CSS) de la pàgina inicial d'autentificació i registre. És la vista relacionada amb el controlador *indexContoller*.

- c_videoconferència

  És la interfície (HTML+CSS) de la pàgina de videoconferència.

- p_perfil

  És la interfície (HTML+CSS) de la pàgina a la que accedeix l'usuari un cop s'autentifica. Aquesta interfície està composta per 4 vistes (es mostra en la fase de disseny) i per això usa varis controladors (un per a cada vista diferent). Hi ha una vista general per a la gestió de converses, enviar missatges..., una vista per a mostrar usuaris i envia'ls-hi peticions d'amistat, una vista per a la gestió d'amistats i creació de peticions múltiples i una última vista per a la gestió de peticions enviades i rebudes.

- índexController

  És el controlador associat a la vista de la interfície d'entrada, entre ell i un fitxer general de JavaScript s'encarreguen de la lògica de la primera de les planes Web. Des del controlador és fan les peticions HTTPS (usant l'objecte d'angular *$http*) per a registrar i autentificar l'usuari que interactua amb la pàgina.

- ConversesController

  És el controlador relacionat amb la interfície de videoconferències. No té peticions HTTP, ja que aquesta pàgina és pràcticament independent respecte el servidor. Al ser la plana de videoconferències es té una instància de la classe *Conversa*, que actua com a pont amb l'API per a la gestió de videoconferències. A través de *Conversa* el controlador envia i rep events de la "plataforma" *ASWRTC* (la classe visible des de fora l'API *ASWRTC* és *ClientWebRTC* i la interfície implementador*ClientWebRTC*).

- amicsConController

  És el controlador associat a una de les vistes de la pàgina ***p_perfil***. El controlador rep i envia missatges de la vista referent a la gestió d'amics i enviament de peticions de conversa múltiples. S'encarrega d'obtenir través d'HTTP el llistat d'amics de l'usuari un cop entra a la

pàgina. Amés per enviar peticions de converses múltiples té una instància de la classe *LlistaContactesConSock*, que li permet poder usar els mètodes de la part de Socket.IO.

- **peticionsConvController**

  És el controlador de la vista per a gestionar les peticions de conversa enviades i rebudes de la interfície ***p_perfil***. Un cop iniciada la pàgina el controlador obté del servidor totes les peticions pendents i les mostra a la vista. Per a la resposta de peticions en temps real també s'usa la classe *LlistaContactesConSock* que alhora usa les classes amb Socket.IO.

- **PerfilConveresController**

  És l'objecte VM ("controlador") per la vista principal de la pàgina ***p_perfil***. Proporciona la lògica per a la selecció de converses, control de videoconferències actives, canvi de les dades del perfil de l'usuari, configuració de trucades i enviament de missatges, fitxers, imatges i trucades entre els usuaris d'una conversa.

  Al igual que els altres controladors de la interfície *p_perfil* obté les dades inicials a través de crides HTTPS al servidor i llavors, per a la gestió dels esdeveniments en temps real delega la feina a la classe *LlistaContactesConSock*.

- **usuarisConroller**

  Controlador per a la vista que mostra els usuaris que pots agregar com amics i des de la qual se'ls i pot enviar peticions d'amistat.

- **amplesBanda**

  Crea l'objecte *amplesBanda* i proporciona una llista de mètodes per interactuar amb ell. *amplesBanda* conte les velocitats de transmissió de pujada i baixada de l'usuari i guarda el seu contingut a la memòria local del navegador.

- **perfilFactory**

  Crea l'objecte ***dadesGenerals*** que es compartit per tots els controladors de la vista *p_perfil*. L'objecte *dadesGenerals* conté tota la part comuna entre tots els controladors.

- **Conversa**

  Classe que fa de pont entre el controlador de la pàgina de videoconferències i la API *ASWRTC*. La classe relaciona la vista amb la llibreria de videoconferències, per fer-ho té una instància de la classe *ClientWebRTC* i implementa la interfície per on es reben els esdeveniments que la videoconferència genera (*ImplementadorClientWebRTC*), l'objecte *ClientWebRTC* s'encarrega de la gestió de la llibreria WebRTC i es comunica amb el la classe *Conversa* cridant els mètodes de la interfície *ImplementadorClientWebRTC* que la classe ha de tenir implementats. Si no s'implementa la interfície *ImplementadorClientWebRTC* llavors nosaltres podrem enviar senyals a la videoconferència però no reaccionarem als missatges que ens envien els altres usuaris.

- **LlistaContactesConSock**

  Classe que fa de pont entre tots els controladors de la pàgina *p_perfil* i la part per a la comunicació amb el servidor en temps real (API ASSIO). La classe crida els mètodes de la classe *SimplySocketIO* (per fer-ho crea una instància de la classe *ConnexioSocketIO* indicant-li que vols que es comporti com a *SimplySocketIO*) per enviar missatges a través de Socket.IO al servidor i altres usuaris, per altre banda quan es reben missatges la classe *SimplySocketIO* crida els mètodes de la interfície *ImplementadorSymplySocketIO*, per tant, per rebre accions

d'altres usuaris que repercuteixen en l'estat de l'usuari local cal implementar els mètodes de *ImplementadorSymplySocketIO*.

- **SimplySocketIO**

  Classe per a la gestió dels missatges de Socket.IO a la part no referent a la videoconferència. La classe té un objecte socket que comunica bidireccionalment la part client amb el servidor i per tant es l'encarregada d'enviar els missatges Socket.IO que la classe *LlistaContactesConSock* li delega. Aquesta classe envia i rep missatges referent a l'estat en que es troben els usuaris local i remots, peticions de converses, missatges de converses, etc.

- **ImplementadorSimplySocketIO**

  Interfície que ha d'implementar tota classe que tingui un objecte *SimplySocketIO*, els mètodes especificats en la interfície són els que cridarà la classe *SimplySocketIO* per informar a la vista de les accions preses per altres usuaris que tenen una relació amb l'usuari local.

- **WebRTCConnection**

  Classe per a la part referent a videoconferències. Aquesta classe implementa els mètodes de la interfície *SubjecteWebSocket* per a que les classes que l'observen (que tenen una instància seva) sàpiguen els mètodes amb els quals la poden cridar; d'altre banda la classe crida els observadors usant els mètodes de la interfície *ObservadorWebSocket*. La classe té un objecte ***socket*** que s'encarrega d'enviar i rebre peticions referents al protocol de senyalització de WebRTC.

- **ConnexioSocketIO**

  Classe principal de l'API ASSIO que permet poder crear connexions *socket* amb el servidor amb diferents comportaments. Quan una classe vol usar Socket.IO crida la classe *ConnexioSocketIO*, tal com fan les classes *ClientWebRTC* I *SimplySocketIO*. Quan es crida *ConnexioSocketIO* pots triar quin namespace usar del servidor. Si uses el namespace 'grup' la classe *ConnexioSocketIO* t'assignarà els objectes *WebRTCConnection* i *MissatgesDeSala*, d'altre banda si uses el namespace 'usuari' t'assignarà els objectes *WebRTCConnection* i *MissatgesIndividuals*.

  D'aquesta manera es vol fer que el diagrama es vegi poc afectat per a la creació de nous namespaces al servidor.

  Amés la classe *ConnexioSocketIO* permet canviar el comportament dels missatges en temps d'execució (*MissatgesDeSala*, *MissatgesIndividuals*) permetent enviar a tots els usuaris d'un grup o només a usuaris individuals.

- **Missatge**

  Interfície que conte els mètodes que han de tenir els diferents comportaments de missatges.

- **MissatgesDeSala**

  Implementa els comportament per a missatges de grup. Permet enviar missatges Socket.IO a tots els usuaris pertanyents a un grup determinat.

- **MissatgesIndividuals**

  Implementa els comportament per a missatges individuals. Permet enviar missatges Socket.IO a un usuari determinat.

- **ClientWebRTC**

  Classe principal de l'API ASWRTC. S'encarrega d'establir una multi conferència entre varis usuaris usant WebRTC, la classe delega la feina a varies classes però ha de permetre crear una multi conferència només interactuant amb ella. Pensant en la generació d'una API aquesta és la classe visible des de fora la plataforma (juntament amb la interfície *ImplementadorClientWebRTC*).

  Per poder complir amb les seves funcionalitats aquesta classe actua com a observador de les classes *MediaStreamLocal*, *Connexió* i *WebRTCConnexion* per tal d'obtenir l'objecte MediaStream a enviar, establir tantes PeerConnections com usuaris hi hagi d'haver a la sala i poder connectar els usuaris amb el protocol de senyalització a través de Socket.IO.

  Quan una classe vol implementar videoconferències ha de tenir una instància d'aquesta classe i implementar els mètodes de la interfície *ImplementadorClientWebRTC* ja que conté tots els mètodes que *ClientWebRTC* crida per notificar els esdeveniments rebuts d'usuaris remots.

- **ImplementadorClientWebRTC**

  Interfície amb tots els mètodes que la classe *ClientWebRTC* utilitza per notificar diferents esdeveniments a les classes que usen la l'API ASWRTC. Una classe que té un objecte *ClientWebRTC* ha d'implementar els mètodes d'aqueta interfície si vol que tot funcioni correctament.

- **ObservadorMediaSreamLocal**

  Conté tots els mètodes que ha d'implementar un objecte que actuï com a observador de la classe *MediaSreamLocal*. Per exemple, la classe *ClientWebRTC* al implementar aquests mètodes permet que la classe *MediaSreamLocal* li pugui notificar els esdeveniments que li sorgeixen.

  Gràcies al patró *observer* la classe *MediaSreamLocal* pot executar mètodes asíncrons i informar dels resultats quan acaben.

- **ObservadorConnexió**

  Conté tots els mètodes que s'han d'implementar si crees almenys un objecte *Connexió*. Un *ClientWebRTC* pot tenir varies connexions, però d'interfície només n'hi ha una ja que els mètodes de d'interfície són a nivell de conversa, no de connexió individual entre un usuari.

- **ObservadorWebSocket**

  Interfície amb els mètodes que podem rebre a través de SOCKET.IO per a la gestió d'usuaris dins la conversa i el protocol de senyalització de WebRTC. Si ens subscrivim a un objecte WebSocket podem rebre notificacions (indicant que un usuari ha entrat la conversa (o n'ha sortit), els missatges SDP d'altres usuaris...) gràcies aquesta interfície.

- **SubjecteMediaStreamLocal**

  Interfície que ha d'implementar la classe observada *MediaStreamLocal*. Els observadors de *MediaStreamLocal* poden parlar amb ella a través d'aquests mètodes fent així que l'observador pugui enviar també dades a l'objecte observat.

- **SubjecteConnexió**

  Interfície que ha d'implementar la classe observada *Connexió* per a que els observadors sàpiguen les funcions que tenen disponibles.

- **SubjecteWebSocket**

  Interfície que ha d'implementar la classe observada *Connexió* per a que els observadors sàpiguen les funcions que tenen disponibles.

- **MediaStreamLocal**

  Classe encarregada de l'objecte MediaStream obtingut dels dispositius locals. La classe s'encarrega de crear l'objecte MediaStream obtinguen una URI única per a cada un dels tracks que recullen la informació dels dispositius locals. La classe permet la creació de l'objecte MediaStream a partir de les característiques que se li marquen, permetent crear un objecte MediaStream amb vídeo i àudio, només vídeo o només àudio, amés en el cas d'obtenir vídeo de la càmera, dona l'opció d'escollir les imatges per segon que es recullen (fps).

- **Connexió**

  Classe que estableix una connexió WebRTC entre dos usuaris. La classe crea un objecte PeerConnection que permet enviar i rebre qualsevol flux multimèdia a l'usuari remot que se li indica, amés estableix també un canal per enviar dades entre l'usuari local i el remot.

  Una multi conferència tindrà tants objectes d'aquesta classe com usuaris remots i participin, d'aquesta manera podem enviar i rebre dades entre cada usuari participant i nosaltres a través d'un canal P2P .

- **ControladorEstats**

  Proporciona a la classe Connexió les estadístiques de com està funcionant la connexió WebRTC amb l'usuari remot. Aquesta classe permet que la classe Connexió delegui la feina d'haver de calcular les estadístiques (feia que el codi de la classe quedes molt llarg).

Patrons utilitzats:

Per a la construcció del diagrama de classes no s'ha usat cap patró de manera estricte, però si que s'han usat les idees de tres patrons GOF; el ***Factory***, l'***Observer*** i l'***Strategy***.

El **patró Factory** (patró creacional) ja venia ideat per a AngularJS i permet crear una classe dedicada únicament a la fabricació d'un objecte per tal de no exposar el procés de creació al client i retornar un objecte a través d'una interfície comuna.

En el nostre diagrama, el patró s'usa en el cas en que tenim múltiples controladors per una sola interfície permetent treure'ls-hi, als controladors, la responsabilitat de creació de certs objectes compartits entre ells.

Gràcies a l'augment de cohesió que proporciona el patró Factory (grau en que les classes estan relacionades) estalvia l'ús repetitiu de mètodes i redueix la comprensió del codi i el treball.

Les dos classes Factory del diagrama de classes són ***PerfilFactory*** i ***AmplesBanda***; **el patró no es veu reflectit al diagrama com hauria de ser, però per ocupar menys espai s'han unit les classes de *Objecte Concret* i *interfície de l'objecte* tot dins la classe Factory.**

El **patró Observer** (patró de comportament) s'usa degut a que hi han classes que poden rebre esdeveniments de les classes que els han instanciat, però també de les classes que instancien.

D'aquesta maner, en el diagrama de classes, quan s'instancia una classe que després t'enviarà esdeveniments s'usa aquest patró.

Aquest fet passa especialment a la classe *ClientWebRTC* que usa classes que li envien esdeveniments, aquestes classes es dibuixen com a **subjectes**, i la classe ClientWebRTC s'apunta com el seu **observador**, d'aquesta manera, quan el subjecte rep un esdeveniment remot avisa als observadors que té apuntats. Per tal de que no hi hagi un alt acoblament entre subjecte i observador es fa una interfície per a cada rol, la interfície pel subjecte indica els mètodes que ha d'implementar el subjecte i que podrà usar l'observador per interactuar-hi, d'altre banda, la interfície de l'observador ha de ser implementada per tota classe que actuï com a observador del subjecte, ja que té tots els mètodes que usa el subjecte per notificar els esdeveniments a l'observador.

En el diagrama de classes de la API ASWRTC tenim:

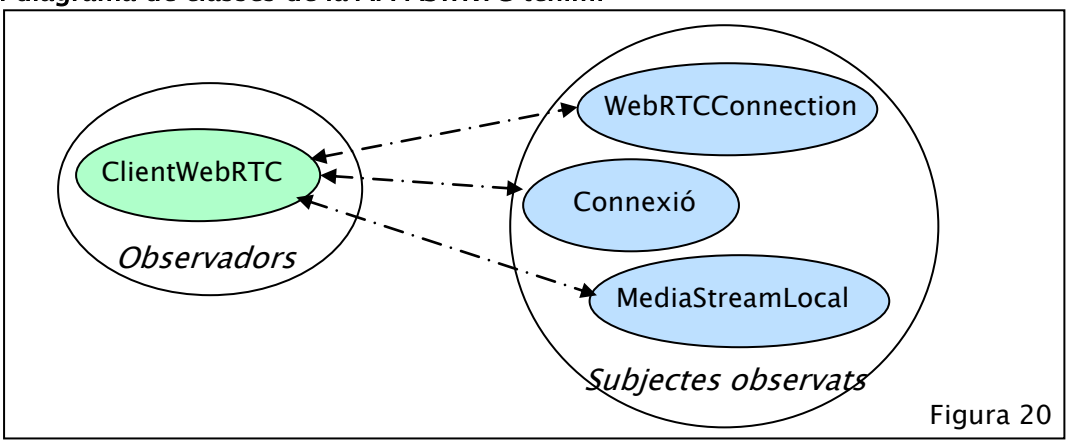


Figura 20

El patró observador dona al diagrama de classes un menor acoblament que el que tindríem si relacionéssim els objectes directament sense interfícies; això es degut a que:

-El subjecte observat només necessita saber de l'observador que implementa la interfície d'observadors.

-Podem afegir tants observadors com vulguem al subjecte observat i no canvia res.

-Els canvis al subjecte observat no afecten als observadors si es respecta la interfície (i viceversa).

-És fàcil poder reutilitzar l'esquema per a altres diagrames.

Partint d'aquestes avantatges, també s'ha usat l'idea del patró observador per a la part visible de les "plataformes" (API's) de videoconferències i Socket.IO que s'han fet. En aquest cas els objectes *ClientWebRTC* i *SimplySocketIO* cada un té relacionada una interfície (*Implementador-ClientWebRTC* i Implementador*SimplySocketIO*) perquè els clients que usen les llibreries sàpiguen els mètodes amb els quals la llibreria els i pot enviar esdeveniments.

Per últim, el **patró Strategy** (patró de comportament) s'ha usat per a representar 2 possibles comportaments que poden tenir les classes *SimplySocketIO* i *WebRTCConnection* alhora d'enviar missatges. D'aquesta manera *SimplySocketIO* i *WebRTCConnection* poden canviar en temps d'execució l'opció d'enviar missatges de manera individual a fer-ho de manera grupal i viceversa.

Aquest doble comportament estava pensat especialment per oferir a un *"namespace"* Socket.IO del servidor diferents comportaments, però amb la separació de *namespaces*, la possibilitat de tenir diferents comportaments ja no es tant útil.

8.1.3. Estudi de la base de dades fase anàlisi

En la fase de requeriments s'han exposat els requeriments de l'aplicació, però no s'ha mirat encara que guardar a la base de dades per poder complir amb totes les funcionalitats. Aquest apartat s'agüés pogut fer en l'estudi de funcionalitats però per tal de deixar per l'anàlisi de requeriments només la part *front-end* s'ha decidit fer la fase de requeriments i anàlisi de la BD tota aquí. Un cop fet això ja s'ha de poder dissenyar el diagrama entitat/relació per la base de dades.

Les dades que es preveuen guardar són les següents:

- Els usuaris que s'han registrat a l'aplicació i les dades del seu perfil.

  Per cada usuari s'ha de guardar: *Codi, NickName, Sexe, Password, Corrreu, Avatar, Data d'últim accés a l'aplicació*.

- Una llista d'amics per a cada usuari, per poder així llistar els usuaris els quals se'ls hi ha enviat una petició d'amistat i l'han acceptat. Això permetrà poder crear converses múltiples.

  Per cada amic s'ha de guardar: *Codi, NickName, Sexe, Password, Corrreu, Avatar, Data d'últim accés a l'aplicació*.

- Totes les converses de l'aplicació i els usuaris que i participen. Això permetrà satisfer les funcionalitats de que cada usuari pugui gestionar les converses en les que participa.

  Per cada conversa s'ha de guardar: *Codi, Usuaris participants, data d'última modificació, llista amb tots els missatges, imatges i arxius intercanviats entre els usuaris*.

- Una llista de peticions pendents entre usuaris.

  Per cada petició s'ha de guardar: *Codi, Usuaris implicats i la resposta dels usuaris a la petició.*

  En aquest cas, un cop la petició ha estat resposta ja es pot esborrar.

- La configuració de les videoconferències d'un usuari.

  Per cada configuració s'ha de guardar: *Codi, Usuari, si vol àudio, si vol vídeo, si permet gravacions.*

- Les ubicacions des d'on s'ha connectat un usuari i les velocitats de transmissió de pujada i baixada que ha donat el test de velocitat.

  Cal guardar una llista amb: *Codi, Usuari i el parell {ubicació,amples de banda}.*

- Per a cada conversa, s'ha de tenir un llistat de tots els missatges, imatges, arxius i trucades emmagatzemat.

  Per cada "event" cal guardar: *Codi, Conversa del esdeveniment, Usuari que l'ha fet i el contingut de l'esdeveniment (missatge, fitxer, imatge, trucada).*

8.1.4. Estudi API Socket.IO fase anàlisi

Del conjunt de dades guardades a la base de dades algunes han de poder ser actualitzades i notificades en temps real pels usuaris que estan en línia. Per exemple, si s'envia un missatge a través del xat d'una conversa la resta d'usuaris de la conversa que estan en línia han de rebre el missatge en el mateix moment. Per a fer això s'ha decidit construir una API bidireccional en Socket.IO que supleix la mancança de que el servidor no pot notificar esdeveniments al client sense que abans el client inicïi la conversa amb ell (problema que té la API REST).

Les dades que han de poder ser notificades en temps real als usuaris remots són:

- Quan l'usuari local els hi envia una petició de conversa.
- Quan l'usuari els hi respon una petició d'amistat que li havien enviat.
- Quan un usuari envia un missatge, trucada, fitxer o imatge en una conversa.
- Si l'usuari no estava en línia i passa a estar-ho (i viceversa).
- La resposta d'una trucada feta per un altre usuari.

De fet, cal que sigui en temps real pràcticament tots els mètodes POST de l'API REST, per això alhora del disseny queden molt pocs mètodes POST a l'API REST i es passen a l'API de Socket.IO.

8.2. Disseny

Aquest apartat està destinat a la part de disseny del sistema. Primer es mostren com s'han dissenyat les interfícies i llavors la lògica del sistema.

Pel que fa la lògica del sistema, per la part client es desenvoluparan els diagrames de seqüència dels diferents casos d'ús, per tal de veure com interaccionen les classes del diagrama de la fase d'anàlisi i trobar els atributs i mètodes que compondran cada classe. Per la part del servidor és dissenyarà la base de dades. Finalment, es mostra el disseny de l'API REST i l'API Socket.IO que permeten la comunicació client-servidor pel primer cas i {client-servidor | servidor-client} pel segon.

8.2.1.Interfícies d'usuari

La Web consta de tres planes, una pel registre i autentificació, una pàgina pel perfil d'un usuari autentificat i una pàgina dedicada a les videoconferències.

Pàgina per la identificació:

Figura 21

Pàgina per la personalitzada pel perfil de l'usuari:

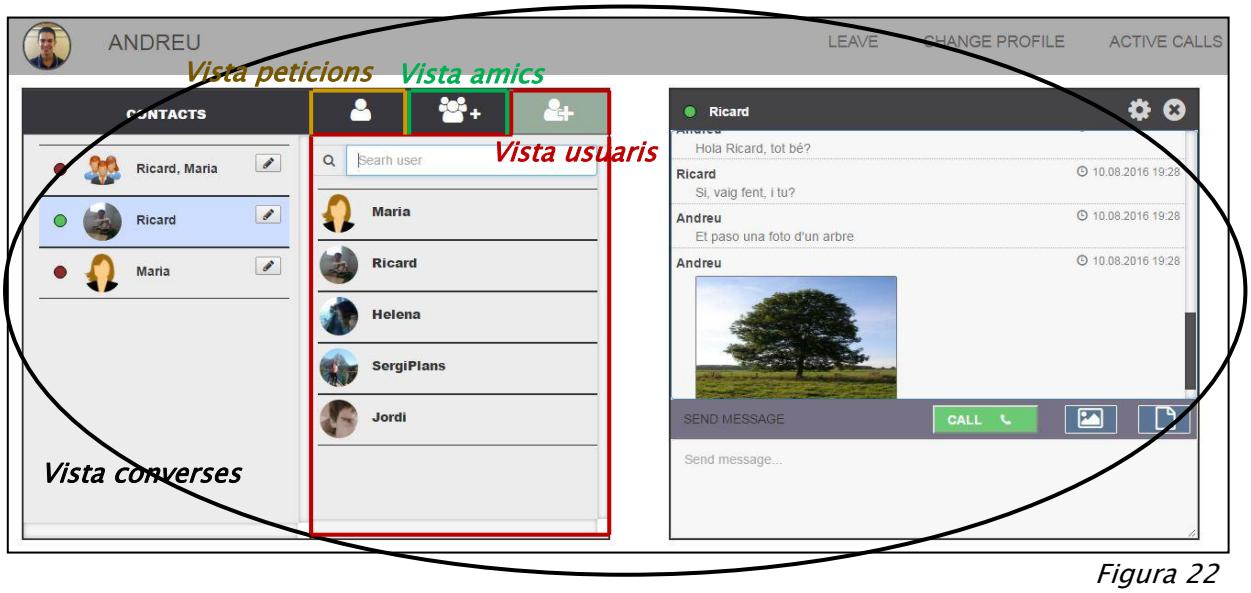


Figura 22

Aquesta plana, a diferència de les altres esta composada per 4 vistes diferents (tal com es mostra en la imatge).

Cada vista és independent a les altres tot i que els controladors relacionats a cada vista comparteixen algunes dades.

Pàgina per a la realització de videoconferències:

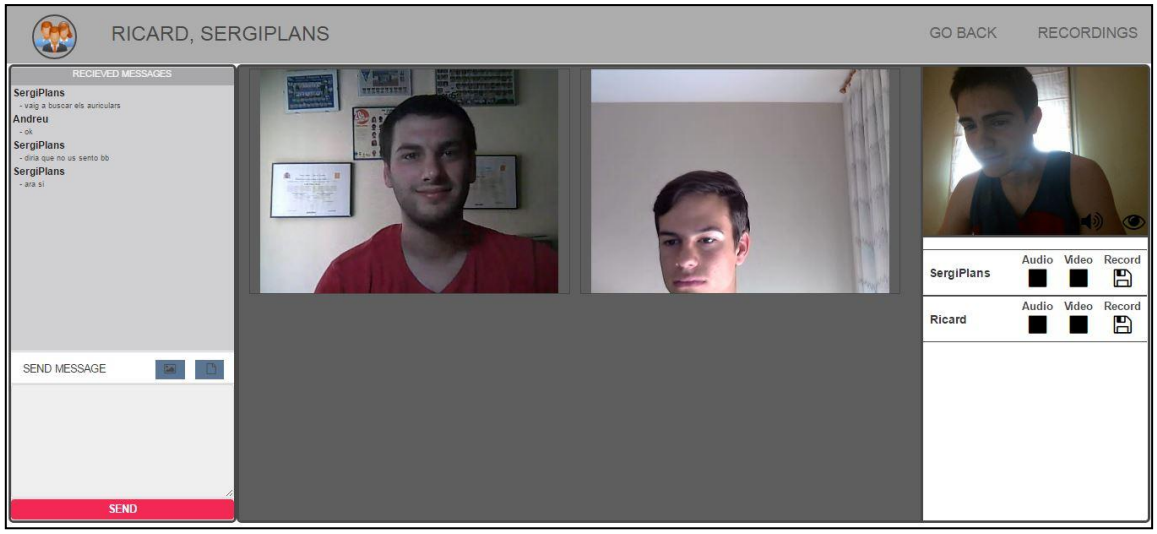


Figura 23

Totes les pàgines tenen contingut desplegable que no es mostra, però sense desplegar res aquest és el que es veu en cada una de les interfícies.

8.2.2.Model de processos

En aquest apartat es descriu com col·laboren els objectes en funció del temps per a cada un dels casos d'ús. El resultat serà un llistat de mètodes que permetran construir el diagrama de classes de disseny.

En aquesta secció, per resumir, es mostren només les classes que intervenen a cada cas d'ús sense especificar com interactuen entre elles. En l'apartat 15.2 (Diagrames de seqüència) d'annexes estan representats els diagrames de seqüència, especificant les classes que intervenen en els casos d'ús en funció del temps i els mètodes que usen.

Només és mostren les interaccions iniciades per l'usuari que van en sentit d'entrada (des de la pantalla cap a la lògica), suposant que per mostrar els resultats es seguirà el procés invers que passarà per les mateixes classes (excepte en l'ús de les API's). Les interaccions entre classes arriben fins a les classes frontera de l'API ASWRTC i fins la classe que usa l'API ASSIO, s'ha fet així per no disminuir la complexitat els diagrames de seqüència i perquè un nou programador no s'ha de preocupar de la implementació interna de les API's ASWRTC i ASSIO.

INTERACCIÓ ENTRE CLASSES PÀGINA INICIAL:

Interacció cas d'us Registrar Usuari:

Vista Índex -> IndexControler -> API REST

Interacció cas d'us Identificar Usuari:

Vista Índex -> IndexControler -> API REST

INTERACCIÓ ENTRE CLASSES DE LA PÀGINA PERSONALITZADA PELS USUARIS :

Interacció cas d'us Modificar perfil

Usuari identificat -> Vista converses -> ConversesController -> API REST

Interacció seqüència cas d'us Gestionar usuaris:

-Llistar usuaris

Usuari identificat -> Vista usuaris -> UsuarisController -> API REST

-Enviar petició amistat

Usuari identificat -> Vista usuaris -> UsuarisController -> LlistaContactesConSock

Interacció cas d'us Gestionar Peticions:

-Llistar Peticions conversa

Usuari identificat -> Vista peticions -> PeticionsConvController -> API REST

-Resposta petició conversa

Usuari identificat -> Vista peticions -> PeticionsConvController -> LlistaContactesConSock

Interacció cas d'us Gestionar amics:

-Llistar amics

Usuari identificat -> Vista amics -> AmicsConvController -> API REST

-Enviar petició conversa

Usuari identificat -> Vista amics -> AmicsConvController -> LlistaContactesConSock

Interacció cas d'ús Enviar event:

UsuariIdentificat->Vista converses->ConversesController->APIREST->LlistaContactesConSock

Interacció cas d'us Llistar converses:

ConversesController -> API REST -> Vista converses

Interacció cas d'us Esborrar Conversa:

Usuari identificat -> Vista converses -> ConversesController -> LlistaContactesConSock

Interacció cas d'us Mostrar trucades actives:

Usuari identificat -> Vista converses -> ConversesController

Interacció cas d'us Ubicar Usuari:

Sistema -> ConversesController -> API REST

Interacció cas d'us Comprovar connectivitats:

Sistema -> ConversesController -> LlistaContactesConSock

INTERACCIÓ ENTRE CLASSES PÀGINA VIDEOCONFERÈNCIA :

Interacció cas d'us Gestionar usuari local o remot:

Usuari trucant->Vista videoconferència->ConversesController->Conversa->ClientWebRTC

Interacció cas d'us Enviar xat videoconferència:

Usuari trucant->Vista videoconferència->ConversesController->Conversa->ClientWebRTC

Interacció cas d'us gestionar VDT:

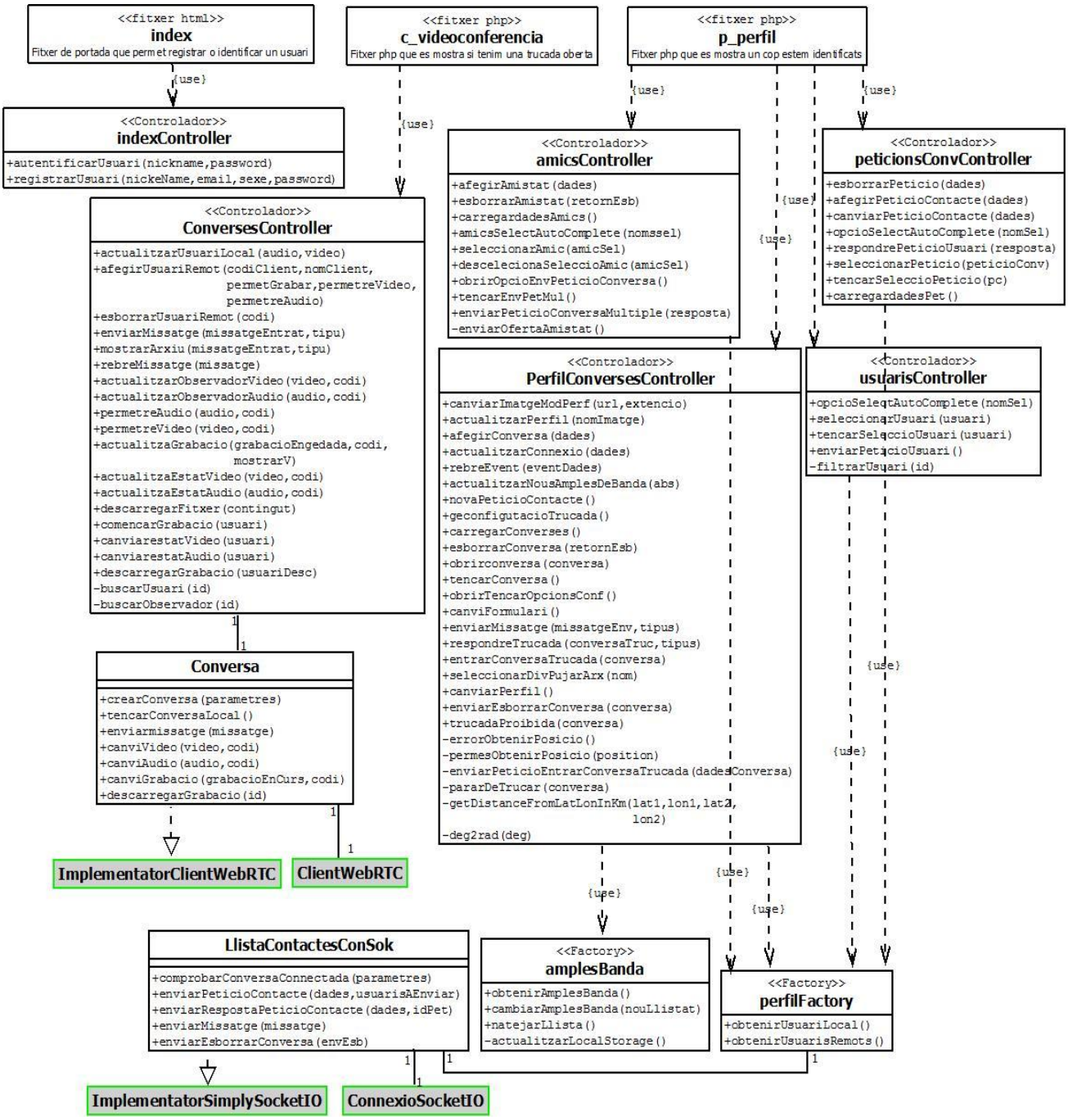
Sistema->Connexió->ControladorEstats->Connexió->ClientWebRTC->WebRTCConnection

8.2.3.Disseny del diagrama de classes

A partir del diagrama de classes d'anàlisi i els diagrames de seqüència de l'apartat 15.2 d'annexes el diagrama de classes de disseny per a la part client és el següent. El diagrama es mostra dividit en 3 parts, la part que composa el *front-end*, l'API ASWRTC i l'API ASSIO.

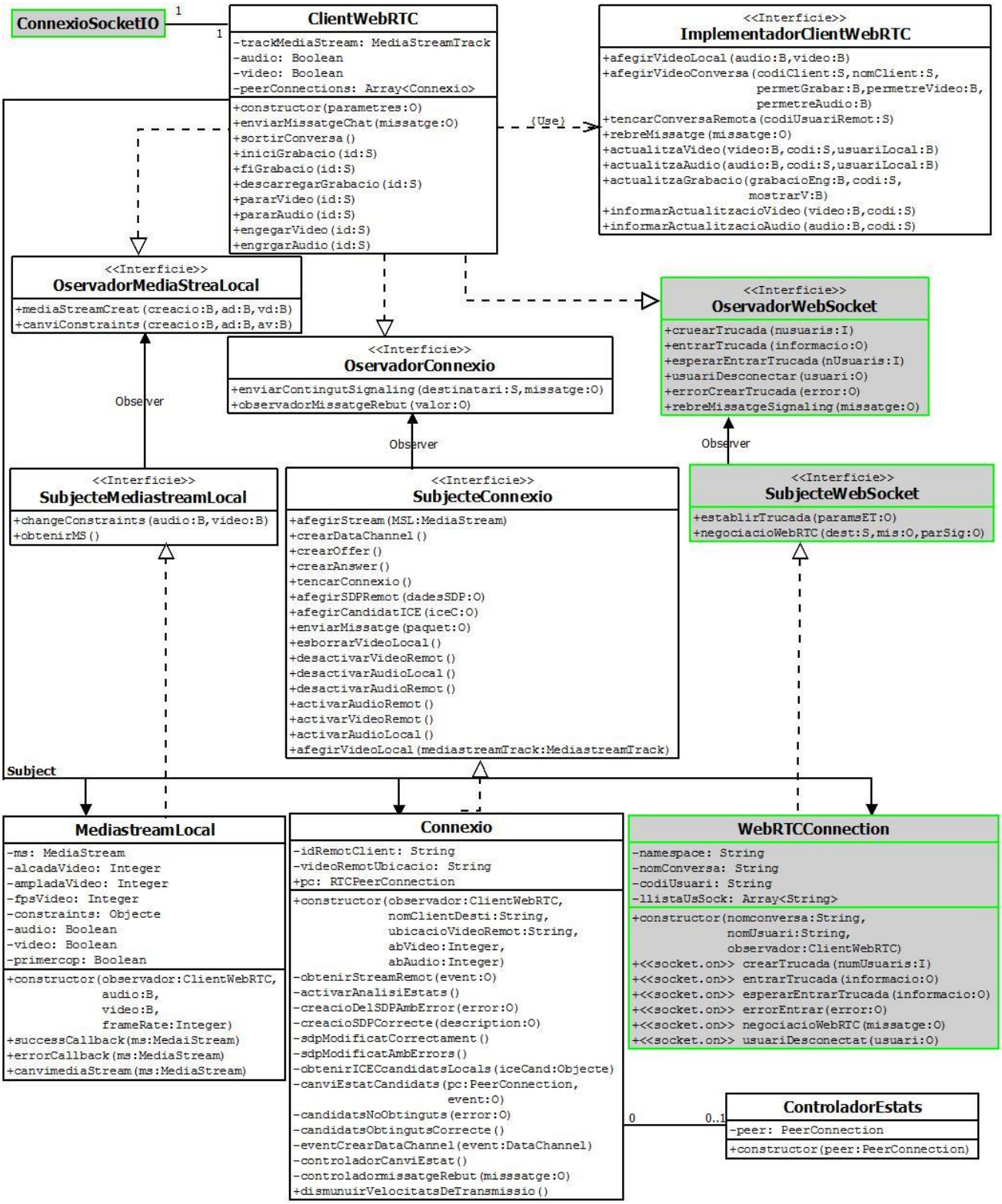
Part del diagrama referent a la part *front-end*:

En el diagrama es mostren en verd les classes frontera de les dos API. Les classes que es mostren en aquesta part són totes les que tenen a veure amb la part d'interfície del sistema.



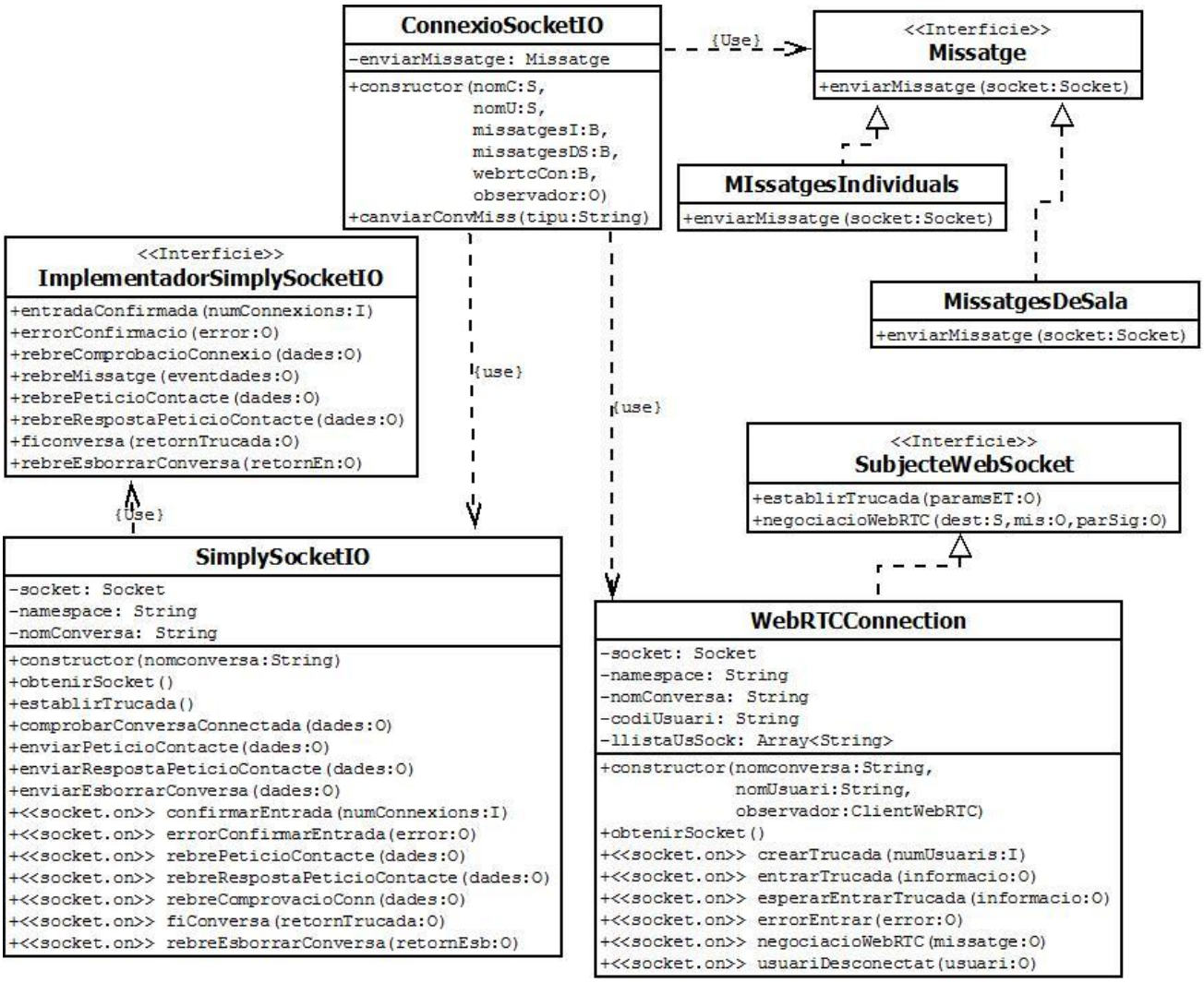
Part del diagrama referent a l'API ASWRT (API sobre WebRTC):

Aquesta es la a part referent a l'API ASWRTC, aquesta part es dependent de l'API ASSIO per a donar suport a la restricció del protocol de senyalització de WebRTC i poder fer la renegociació de velocitats de transmissió en temps real. Els elements d'aquesta segona part que comuniquen amb la tercera estan assenyalats en verd.



Part del diagrama referent a l'API ASWRT (API sobre WebRTC):

Aquesta és la part que permet la comunicació entre client i servidor a través d'un canal bidireccional implementat en Socket.IO. Aquest apartat proporciona a les demés classes la possibilitat d'intercanviar dades en temps real entre usuaris a través d'un canal que passa pel port *8080* del servidor SCS.



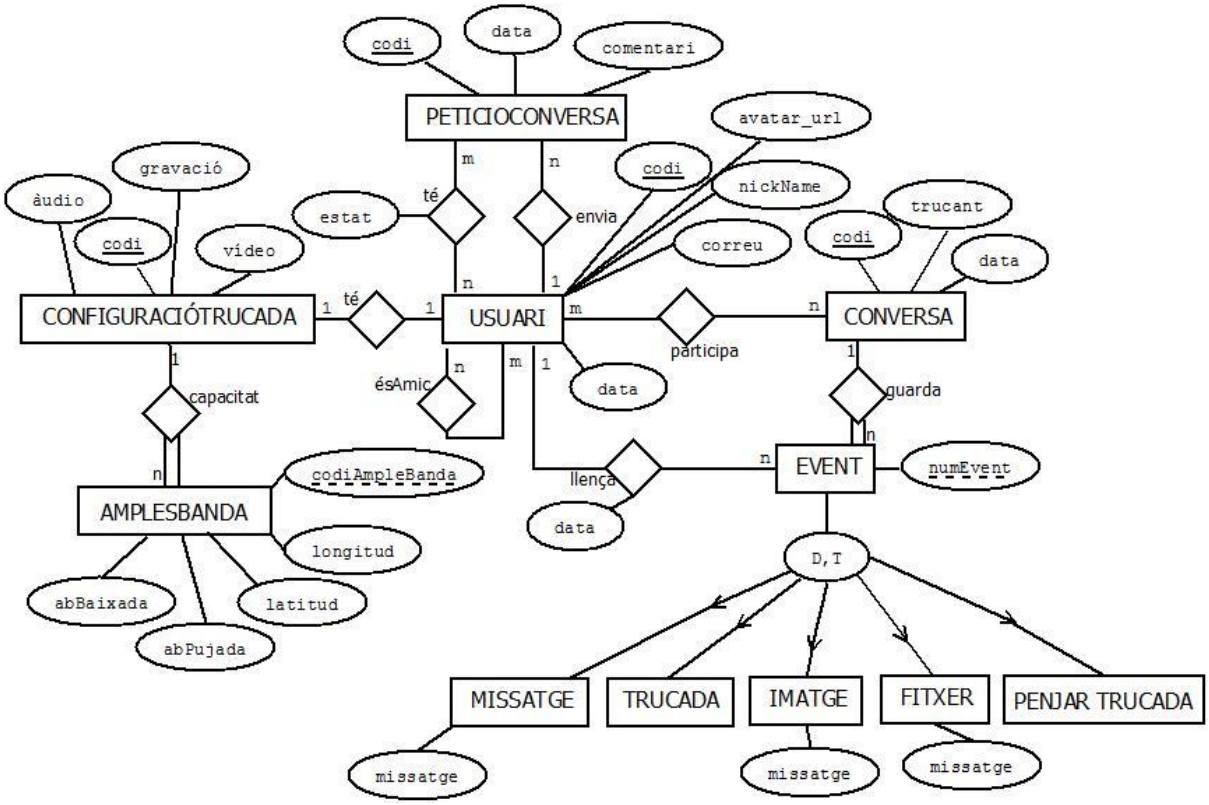
El disseny de la part client acaba aquí, ara es mostrarà del disseny realitzat a la part del servidor i les API's, tant REST, com Socket.IO, que se li proporciona al client per a que pugui interactuar amb la base de dades i altres usuaris.

8.2.4.Disseny de la base de dades

Per a dissenyar la BD, a partir dels requisits de dades a guardar, s'ha realitzat un esquema de la base de dades amb el model entitat/relació i posteriorment, un cop escollit el SGBD (sistema gestor de la base de dades) s'ha dissenyat l'esquema lògic amb el model relacional.

8.2.4.1. Model conceptual de la base de dades

El disseny de l'esquema conceptual de la BD , utilitzant el **model entitat/relació** és el següent:



La funcionalitat de les taules és la que segueix:

Usuari: És la taula principal de l'esquema, cada entitat d'aquesta taula és un usuari registrat. Cada usuari té una configuració de trucada personalitzada i una llista d'amistats, converses en les que participa i peticions de conversa en les que i pertany i estan pendents de resposta.

Petició conversa: Una petició de conversa està relacionada amb el conjunt d'usuaris que participaran a la conversa (si s'accepta); cada usuari participant pot tenir la petició en estat d'acceptada (ha rebut la petició i l'ha acceptat), enviada (ha sigut ell qui ha creat la petició) o pendent (encara no ha respost).

Les entitats d'aquesta taula són temporals, si només un dels *n* participants a la petició la rebutja automàticament es borra la petició pels demés usuaris, altrament, si tots l'accepten la petició de conversa passa a la taula de Conversa.

Per a millorar el sistema de peticions la petició també es guarda l'usuari que l'ha creat, i el comentari que ha escrit per informar als demés de l'objectiu de la petició.

Conversa: Uneix un grup d'usuaris perquè es puguin enviar missatges entre ells. Una conversa té *n* usuaris participants i una llista de tots els esdeveniments que s'han enviat dins del grup.

Configuració trucada: Representa les característiques que un usuari té per a la videoconferència, a la configuració es guarda si l'usuari té permès l'àudio, vídeo i gravació, amés es guarda una llista dels amples de banda segons el punt on es troben.

Amples banda: Cada entitat representa una xarxa diferent en la que s'ha connectat l'usuari. Com que la @IP que usa l'usuari no permet identificar la xarxa, es guarda el punt a on estava geocalitzat l'usuari i la qualitat de la xarxa en la que estava. Aquesta taula permet no recalcular l'ample de banda de les xarxes en les que l'usuari ja hi havia accedit.

Event: És la superclasse dels possibles esdeveniments que poden enviar-se els usuaris d'una conversa. Un esdeveniment pot ser un missatge de text, una imatge, compartir un arxiu, enviar una petició de videoconferència o penjar-la. Per cada esdeveniment es guarda la referència a l'usuari que l'ha creat.

8.2.4.2. Model lògic de la base de dades

En aquesta secció es fa la conversió del model entitat/relació al model lògic depenent del SGBD. Com que la base de dades està gestionada amb MySQL l'esquema lògic es representa amb el **model relacional**.

Pel disseny de l'esquema lògic s'ha seguit el patró explicat a l'assignatura de *Bases De Dades*, molt semblant al de l'enllaç.

Primer es passa el model entitat/relació al següent llistat de taules i relacions:

PeticióConversa (codi,data,comentari)

Usuari (codi,nickName,correu,sexe,password,data,avatar_url)

Conversa (codi,trucant,data)

ConfiguracióTrucada (codi,video,audio,gravacio,codiUsuari)

AmpleBanda (codiAmpleBanda,codiConfTr,latitud,longitud,abBaixada,abPujada)

Event (numEvent,codiConversa,data,contingut,missatge,codiUsuariCreador)

relacio_amic (codiUsuari1, codiUsuari2)

relacio_conversaUsuari (codiConversa, codiUsuari)

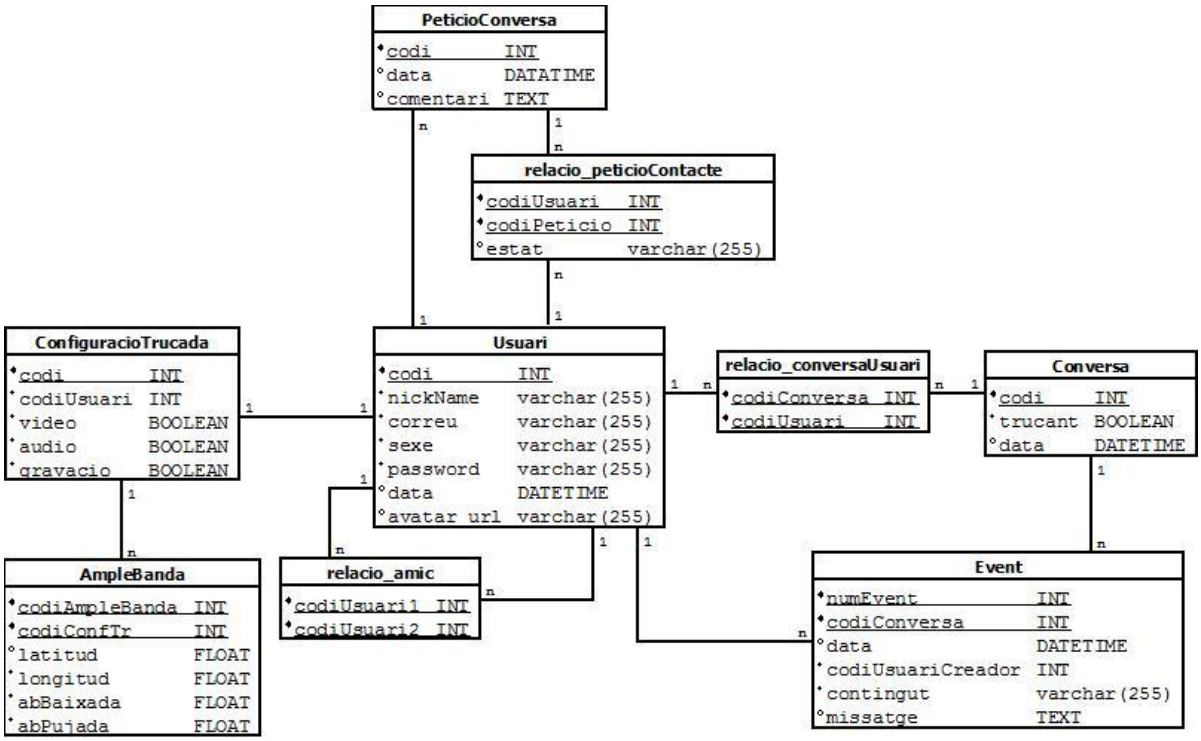
relacio_peticioContacte (codiUsuari, codiPeticio, estat)

El disseny lògic a seguit els passos marcats pel patró i l'única decisió que s'ha pres ha estat en la transformació de les classes amb herència, en aquest cas el patró permet diferents camins per a transformar una especificació de tipus **Disjunta i Total**. En el nostre cas s'ha fet una sola taula per la superclasse i s'han afegit tots els atributs de les subclasses, la clau principal de la nova taula és la de la superclasse i s'ha afegit un atribut *contingut* que indica quina especialització té (al haver tret les subclasses aquest atribut indica a quina subclasse pertany cada entitat).

Aquesta opció és valida per totes les especialitzacions disjuntes; el problema que presenta és que si els camps de les subclasses són molt variats quedaran molts camps nuls, ja que per cada entitat només cal tenir els atributs referents a la subclasse que indica l'atribut *contingut*.

En el nostre cas, s'ha reutilitzat el nom dels atributs a les subclasses per mitigar aquest problema.

Un cop conegudes totes les classes i les restriccions de relació entre elles es dibuixa el model entitat relació:



8.2.4.3. Interfície per accedir a la base de dades

A partir del disseny físic de la base de dades s'ha construït una API REST ubicada a *https://kidd.udg.edu/8080* que permet a la part client del sistema interactuar amb el servidor. L'API permet realitzar les accions GET, PUT, POST i DELETE [33] indicades a la capçalera de les peticions HTTPS i el contingut de les peticions es parceja amb JSON.

El conjunt de crides que formen la API REST estan classificades segons el contingut de la BD que es vol obtenir i dintre d'aquest grup es classifiquen per les accions que es duen a terme. Basant-se en això **la llista de mètodes de la API REST és el següent:**

Usuari

POST

REGISRAR-SE: **POST /usuari/registrar**

Crea un nou usuari i li assigna una configuració de trucada per defecte

-- OMPLENAR FORMULARI:

nickname = *"Nom usuari"*

correu = *"Correu Usuari"*

sexe = *"Gènere usuari"*

password = *"Contrasenya usuari"*

IDENTIFICAR-SE: **POST /usuari/auth**

Verifica si el formulari d'identificació de l'usuari és correcte

-- OMPLENAR FORMULARI:

nickname = *"Nom usuari"*

password = *"Contrasenya usuari"*

OBTENIR USUARIS REMOTS: **POST /usuari/{id}**

Obté els *limitUsuaris* primers usuaris que tenen un identificador diferent a *id* i contenen la cadena *seqChar* en el seu nom.

-- OMPLENAR FORMULARI:

limitUsuaris= *"Numero d'usuaris que es volen obtenir"*

seqChar = *"Seqüència de caràcters que han de tenir el nom de l'usuari"*

AFEGIR PETICIÓ CONVERSA: **POST /usuari/{id}/peticioContacte**

Crea un petició de conversa per l'usuari identificat amb *id*.

-- OMPLENAR FORMULARI:

comentari= *"Missatge que l'usuari adjunta amb la sol·licitud de conversa"*

PUT

MODIFICAR DADES USUARI: **PUT /usuari/{id}**

Modifica les dades d'un usuari

-- OMPLENAR FORMULARI:

nickname = *"Nou nom de l'usuari"*

correu = *"Nou correu de l'usuari"*

avatar_url = *"Direcció a on s'ha guardat la nova foto de perfil"*

GET

OBTENIR DADES USUARI: **GET /usuari/{id}**

Obté les dades personals de l'usuari amb codi=*id*, obté (codi, correu, nickName, sexe, avatar_url i data)

OBTENIR CONFIGURACIÓ TRUCADA USUARI: **GET /usuari/{id}/configutacioTrucada**

Retorna les característiques de trucada de l'usuari (codi, video, audio, grabacio) i la llista de posicions i amples de banda que ha usat al llarg del temps.

OBTENIR PETICIONS CONVERSA: **GET /usuari/{id}/peticionsConversa**

Retorna la llista de peticions que involucren l'usuari amb codi=*id* i l'estat en que les té l'usuari (estat ∈ {acceptada,enviada,pendent}).

OBTENIR AMISTATS : **GET /usuari/{id}/amics**

Obté les dades (codi, correu nickName, sexe, avatar_url i data) per a cada amic de l'usuari amb codi=*id*.

OBTENIR CONVERSES: **GET /usuari/{id}/converses**

Obté la llista de converses i esdeveniments de les converses en les que pertany l'usuari identificat amb *id*.

Configuració trucada

POST

AFEGIR AMPLE DE BANDA CALCULAT: **POST /configuracioTrucada/{id}/ampleBanda**

Afegeix unes noves coordenades i amples de banda per a la configuració *id*

-- OMPLENAR FORMULARI:

latitud = *"Latitud en que es troba l'usuari"*

longitud = *"Longitud en que es troba l'usuari"*

abBaixada = *"Ample de banda de baixada de la xarxa"*

abPujada = *" Ample de banda de pujada de la xarxa"*

PUT

MODIFICAR DADES CONFIGURACIÓ TRUDADA: **PUT /configuracioTrucada/{id}**

Modifica les característiques de videoconferència

-- OMPLENAR FORMULARI:

video = *" Cert si s'envia senyal de vídeo en els videoconferències, altrament fals"*

audio = *" Cert si s'envia senyal d'àudio en els videoconferències, altrament fals"*

grabacio = *"Cert si es permet la gravació en els videoconferències, altrament fals"*

Conversa

POST

OBTENIR EVENTS CONVERSA: **POST /conversa/{id}/events**

Retorna els esdeveniments que ordenats de més a menys recent ocupen les posicions entre "*inici*" i "*fi*".

-- OMPLENAR FORMULARI:

inici = *"Posició que ocupa en la taula ordenada la primera entitat que s'agafa"*

fi = *"Posició que ocupa en la taula ordenada l'última entitat que s'agafa"*

En la API REST hi han alguns mètodes destinats únicament a recuperar dades que estan considerats com a accions POST. Això s'ha fet en els casos en que es vol especificar característiques que han de tenir les dades de retorn, en aquests casos és necessari passar un conjunt de paràmetres i per això no s'usa l'acció GET (que seria lo més normal alhora de consultar dades).

Deixant de banda aquesta excepció la resta de crides van acord amb el que l'acció HTTPS especifica, POST per crear dades, GET per consultar i PUT per modificar.

8.2.5.API Socket.IO

El problema dels missatges HTTPS per a fer peticions a la API REST és que si es fa una petició per afegir o modificar dades, la resta d'usuaris que es veuen afectats per la actualització de les dades no se n'adonen fins que decideixen fer una petició GET.

Per tal de que les actualitzacions de la base de dades siguin actualitzades en temps real per a tots els usuaris afectats, cada usuari que esta en línia estableix un canal Socket.IO bidireccional amb el servidor. D'aquesta manera, quan el servidor rep una actualització que afecta a un usuari que esta en línia li pot enviar l'actualització sense que sigui el client qui li hagi de demanar.

L'API Socket.IO construïda es divideix en dos *namespaces*, un *namespace* anomenat **usuari** per a la gestió de les dades persistents dels clients i un *namespace* **grup** destinat a la gestió de videoconferències.

Un *namespaces* és com una classe que engloba un conjunt de mètodes, el seu ús permet classificar en grups els missatges Socket.IO segons determinades característiques. En el nostre cas la raó per la que s'han separat els missatges en dos *namespaces* és perquè una és exclusiva per la part de videoconferències i l'altre és per la resta de pàgines del sistema que interactuen amb la interfície de l'usuari.

El *namespace* **usuari** es troba a la direcció *https://kidd.udg.edu:8080/usuari* i necessita estar en el mateix servidor que conté la base de dades degut a que interactua amb ella. El *namespace* **grup** està ubicat a *https://kidd.udg.edu:8080/grup*, conté tots es mètodes per al protocol de senyalització WebRTC i no té dependència amb cap altre funcionalitat del servidor.

Els mètodes Socket.IO per cada grup poden ser (pel que fa el servidor) per rebre dades dels clients o per envia'ls-hi. La llista de mètodes vista des del servidor són els següents:

API SOCKET.IO NAMESPACE USUARI

- **crear_entrar** (rebuda)

  Funcionalitat: Afegeix una connexió *socket* entre el servidor i el client.

  Paràmetres:
  nomConversa = *"identificador de l'usuari que estableix el canal"*

  Resposta: **confirmarEntrada** (resposta a l'usuari)

  Funcionalitat: Confirma que s'ha establert un *socket* entre client i servidor

  Paràmetres:
  numConnexions= *"total de connexions actuals que hi ha al servidor"*

  errorConfirmarEntrada (resposta a l'usuari)

  Funcionalitat: Notifica que hi han masses usuaris i l'usuari ha de sortir

  Paràmetres:
  error = *"Missatge d'error que s'envia si hi ha mes de 300000 connexions actives"*

- **comprobarConnectivitat** (rebuda)

  Funcionalitat: Comprova per una conversa si tots els usuaris estan en línia.

  Paràmetres:

  codiConv= *"codi de la conversa analitzada "*

  codiSol= *"Codi de l'usuari que fa la petició"*

  llistaIdentificadors = *"Codi de la resta d'usuaris que pertanyen a la conversa"*

  Resposta: **rebreComprovacioConn** (envia a tots els usuaris de *llistaIdentificadors+codiSol*)

  Funcionalitat: Notifica si tots els usuaris de *codiConv* estan en línia

  Paràmetres:

  codiConv = *"codi de la conversa analitzada"*

  resultat= *"Cert si tots estan en línia, altrament fals"*

- **enviarMissatgeIndividual** (rebuda)

  Funcionalitat: Guarda el missatge enviat de la conversa a la BD i el reenvia a tots els usuaris que estan en línia.

  Paràmetres:

  iddestinataris= *"llista d'usuaris remots que participen a la conversa"*

  event= *" tipus d'event (missatge, arxiu, trucada...) i contingut"*

  codiConv= *"Codi de la conversa a la que s'envia l'esdeveniment"*

  enviador= *"Dades de l'usuari que envia l'esdeveniment"*

  Resposta: **rebreMissatge** (envia a tots els usuaris de *iddestinataris+ enviador.codi*)

  Funcionalitat:Informa a l'usuari de que s'ha enviat un missatge a la conversa

  Paràmetres:

  codiConv = *"Codi de la conversa"*

  enviador = *"Dades de l'usuari que envia l'esdeveniment"*

  event = *"Contingut i tipus d'esdeveniment enviat"*

- **enviarPeticioContacte** (rebuda)

  Funcionalitat: Guarda una petició a la BD i la reenvia a tots els usuaris involucrats.

  Paràmetres:

  destinataris= *"Dades dels usuaris remots als qui s'envia la petició"*

  enviador= *"Dades de l'usuari que envia la petició de conversa"*

  codi= *"Codi de la petició guardada prèviament a la BD"*

  missatge= *"Contingut del missatge enviat adjunt amb la petició"*

  Resposta: **rebrePeticioContacte** (envia a tots els usuaris de *destinataris+enviador*)

  Funcionalitat: Informa a l'usuari de la nova petició de conversa

  Paràmetres:

  participants = *"Dades dels usuaris remots als qui s'ha enviat la petició"*

  enviador= *"Dades de l'usuari que ha enviat la petició de conversa"*

  idp= *"Codi que identifica la petició"*

  missatge= *" Contingut del missatge enviat adjunt amb la petició "*

  statusCode= *"200 si la petició de conversa s'ha guardat bé a la BD"*

- **enviarRespostaPeticioContacte** (rebuda)

  Funcionalitat: Guarda la resposta de l'usuari per la petició a la BD i informa als demés usuaris involucrats de la resposta

  Paràmetres:

  enviador= "*Dades de l'usuari que envia la resposta"*

  destinataris= *" Dades dels usuaris remots que pertanyen a la petició"*

  codi= *"Codi que identifica la petició"*

  resposta= *"Cert si accepta la petició, fals altrament"*

  Resposta: **rebreRespostaPeticioContacte** (envia a tots els usuaris de la petició)

  Funcionalitat: Informa a l'usuari de que hi ha una resposta de la petició

  Paràmetres:

  idp= *" Codi que identifica la petició"*

  participants = *"Dades dels usuaris remots que pertanyen a la petició"*

  acceptat = *"Cert si ha acceptat la petició, altrament fals"*

  codiConversa = *"Si tots han acceptat conté el codi de la nova conversa creada, sinó val -1"*

- **esborrarConversa** (rebuda)

  Funcionalitat: Esborra de la base de dades la conversa demanda i ho notifica als usuaris afectats

  Paràmetres:

  iddestinataris= "*Identificadors dels usuaris remots afectats"*

  idenviador= *"identificador de l'usuari que ha esborrat la conversa"*

  codiConv= *"Codi que identifica la conversa"*

  Resposta: **rebreRespostaPeticioContacte** (envia a tots els usuaris de la conversa)

  Funcionalitat: Informa a l'usuari que ha d'esborrar la conversa

  Paràmetres:

  iddestinataris= *"Identificadors dels usuaris afectats"*

  idenviador= *"identificador de l'usuari que ha esborrat la conversa"*

  codiConv= *"Identificador de la conversa esborrada"*

API SOCKET.IO NAMESPACE GRUP

- **crear_entrar** (rebuda)

  Funcionalitat: Afegeix una connexió *socket* entre servidor i client dins del grup *nomConversa* i avisa als usuaris ja presents al grup perquè iniciïn el protocol de senyalització enviant un *offer* a l'usuari *codiUsuari*.

  Paràmetres:

  codiUsuari= *"identificador de l'usuari que estableix el canal"*

  nomConversa = *"nom del grup (codi de conversa) de la videoconferència"*

  parametres = *"objecte que indica les característiques en que l'usuari entra"*

  Resposta: **crearTrucada** (resposta a l'usuari)

  Funcionalitat: Avisa a l'usuari que és el primer d'entrar al grup

  Paràmetres:

  numClients= *"total d'usuaris que estan en una videoconferència"*

  entrarTrucada (resposta a la resta d'usuari que ja estaven al grup)

  Funcionalitat: Avisa al client que ha entrat un nou usuari al grup

  Paràmetres:

  numeroUsuaris = *"Numero d'usuaris realitzant videoconferències"*

  sockeId = *"Identificador del socket que se li ha assignat al nou usuari"*

  codiClient= *"Identificador del nou usuari"*

  parametres = *"característiques del nou usuari del grup"*

  esperarEntrarTrucada (resposta a l'usuari)

  Funcionalitat: Avisa a l'usuari que ja hi havia gent al grup i li diu qui hi havia

  Paràmetres:

  numeroUsuaris =*"Numero d'usuaris que ja estaven a la videoconferència"*

  clients= *"Codis i sockets dels usuaris ja presents"*

  errorEntrar (resposta a l'usuari)

  Funcionalitat: Avisa a l'usuari que ja no pot entrar al grup

  Paràmetres:

  error = *"Missatge d'error per si hi ha més de 10 usuaris al grup"*

- **enviarMissatgeSala** (rebuda)

  Funcionalitat: Reenvia un missatge de xat.

  Paràmetres:

  destinatari = *"grup al que s'envia el missatge"*

  enviador= *"nom de l'usuari que envia el missatge"*

  data= *"contingut del missatge"*

  Resposta: **rebreMissatge** (resposta al grup d'usuaris pertanyents a *destinatari*)

  Funcionalitat: Envia el missatge al grup *destinatari*

  Paràmetres:

  enviador= *"nom de l'usuari que envia el missatge"*

  data= *"total de connexions actuals que hi ha al servidor"*

- **negociacioWebRTC** (rebuda)

  Funcionalitat: Reenvia un missatge del protocol de senyalització WebRTC a l'usuari remot.

  Paràmetres:

  destinatari = *"usuari destinatari del missatge de signaling"*

  enviador= *"identificador de l'usuari que envia el missatge de signaling"*

  data= *"objecte de tipus OFFER, ANSWER o CANDIDATE"*

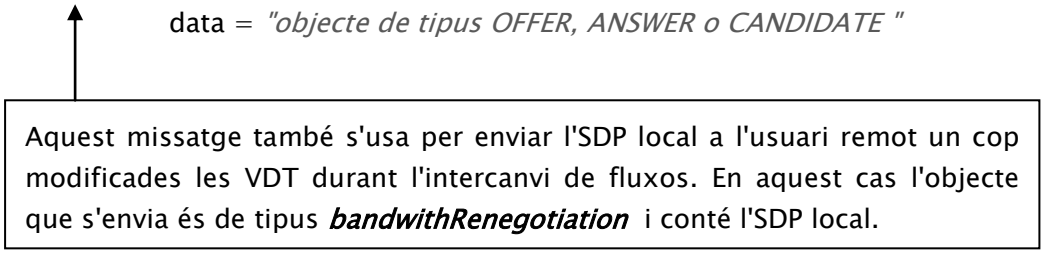
  Resposta: **confirmarEntrada** (resposta a l'usuari *destinatari*)

  Funcionalitat: Envia el missatge de *signaling* al *destinatari*

  Paràmetres:

  enviador= *"codi de l'usuari que envia el missatge"*

  data = *"objecte de tipus OFFER, ANSWER o CANDIDATE "*



- **disconnect** (rebuda)

  Funcionalitat: Informar als demes usuaris i al gestor de que l'usuari abandona la trucada

  Resposta: **usuariDesconectat** (envia per a tots els usuaris del grup)

  Funcionalitat: Informa al client de que l'usuari ha abandonat la videoconf.

  Paràmetres: codiUsuari = *"codi de l'usuari que abandona la trucada"*

  fiConversa (envia al socket del *namespace* usuari)

  Funcionalitat: Informa al socket del *namespace* usuari que l'usuari ha sortit

  Paràmetres:

  codiConv = *"Codi de la conversa en la que es fa la videoconferència"*

  numeroVius = *"Numero d'usuaris que encara estan a la videoconferència"*

A part d'aquest llistat de mètodes del *namespace* grup, **un cop establerta una connexió WebRTC**, tots els missatges referents al control de fluxos que s'envien per a notificar a un usuari remot que li hem parat/engegat el vídeo o àudio. O avisar-lo que em parat/engegat els nostres fluxos que li enviàvem. Tots aquets **missatges s'envien a través del canal de dades (DataChannel) per tal d'aprofitar el canal directe i no fer passar els missatges pel servidor.**

En aquest cas, s'han definit deu tipus de missatges que es poden enviar pel *DataChannel* per tal de notificar d'un esdeveniment a l'altre per. Aquests deu missatges són:

- **pararVideoLocal**

  Notifica a l'usuari de l'altre banda del *peer* que hem parat el nostre vídeo perquè l'esborri dels vídeos que mostra (si no es mostra el vídeo negre).

- **pararAudioLocal**

  Notifica a l'usuari de l'altre banda del *peer* que hem parat el nostre àudio.

- **pararVideoRemot**

  Notifica a l'altre *peer* que em parat el seu flux de vídeo que rebíem. Gràcies a aquest missatge l'altre *peer* sap quins altres usuaris veuen el seu vídeo.

- **pararAudioRemot**

  Notifica a l'altre peer que em parat el seu flux d'àudio que rebíem. Gràcies a aquest missatge l'altre peer sap que aquest usuari ja no escolta la seva veu.

- **engegarVideoLocal**

  Notifica a l'altre peer que a partir d'ara li enviarem el nostre vídeo.

- **engegarAudioLocal**

  Notifica a l'altre peer que a partir d'ara escoltarà el nostre àudio.

- **engegarVideoRemot**

  Notifica a l'usuari de l'altre banda del peer que ara tornem a veure el seu vídeo, per tant ens pot afegir a la llista d'usuaris que veuen el seu vídeo.

- **engegarAudioRemot**

  Notifica a l'altre peer que tornem a escoltar el seu àudio i ens ha d'afegir a la llista d'usuaris de la conversa que l'escolten.

Tots els missatges entre usuaris d'una conversa un cop acabat el procés de senyalització s'envien a través del *DataChannel* usant aquest protocol definit. Però hi ha un missatge que no s'envia a través d'aquest canal, el de reducció de VDT's per a converses inestables.

Això s'ha decidit així perquè quan un usuari demana la reducció de VDT és perquè s'estan perdent molts paquets i el *PeerConnection* està funcionant malament. Si enviem el nou SDP local amb les VDT modificades a través del *DataChannel* del *PeerConnection*, el missatge pot tardar molt en arribar, ja que el *PeerConnection* està saturat. Per això, el missatge de tipus ***bandwithRenegotiation*** s'envia a través de Socket.IO passant pel servidor.

9. Tractament de les dades de caràcter personal

Degut al fet a que l'aplicació gestiona dades de caràcter personal identificatives (nom, cognoms, adreça de correu, etc.) i es guarda tot tipus de dades envides entre usuaris, cosa que fa que puguin contenir qualsevol altre tipus de dades de caràcter personal, ha estat necessari tenir en compte el conjunt de normes jurídiques ordenades per la llei orgànica 15/1999 (**Llei de protecció de dades de caràcter personal**) i els seus posteriors decrets.

El tipus de dades i finalitat d'aquestes, que es guarden en la base de dades (fitxer de titularitat privada) que conté les DCP (dades de caràcter personal) no ha estat registrat a **l'agencia espanyola de protecció de dades** degut el fet de que encara no és un aplicació oberta al públic i es podria considerar com un fitxer d'activitats exclusivament personals.

Tot i així, si que s'han tingut en compte les lleis de la LOPD (llei de protecció de dades de caràcter personal), pensant en una posterior publicació de l'aplicació, i per tal de permetre testejar actualment l'aplicació a persones diferents a jo.

Les dades personals que es guarden d'un usuari són:

-Nom usuari.

-Contrasenya.

-Adreça de correu electrònic.

-Direcció a on està ubicada del servidor la imatge de perfil.

-Missatges enviats i rebuts en les converses amb altres usuaris.

-Direcció de on estan ubicats del servidor els arxius i imatges que l'usuari ha enviat o rebut.

Totes aquestes dades es guarden de manera xifrada a la base de dades, l'única dada relacionada amb un usuari que no es guarda de manera xifrada són les estadístiques d'ample de banda, ja que les dades que serveixen únicament per finalitats estadístiques no son regulades per la LOPD.

Seguidament es mostren els articles de la LOPD que es compleixen en el projecte.

PRINCIPIS DE PROTECCIÓ

Respecte els principis de protecció de la LOPD, es tenen en compte les següents característiques sobre els següents articles:

Article 4- Qualitat de dades

Les dades recollides només són utilitzades pel seu tractament i no s'usen per finalitats incompatibles a aquelles per les quals han estat recollides.

En cas de dades inexactes es dona a l'usuari la possibilitat de modificar el seu perfil, per tal de que pugui cancel·lar les antigues dades i substituir-les per les noves.

És prohibeix la recollida de dades per mitjanats fraudulents, encriptant tant el contingut de la base de dades, com els missatges que s'envien per la xarxa entre els usuaris i la BD.

Article 5- Dret d'informació i recollida de dades

Els interessats als quals se'ls sol·licita dades de caràcter personal són informats del següent:

- De l'existència d'un fitxer de dades de caràcter personal

-Del caràcter obligatori o no de les preguntes plantejades en el registre

-De la conseqüència de l'obtenció de dades o la negativa a subministrar-la

-De la possibilitat d'executar els drets d'accés, rectificació, cancel·lació o oposició.

-De la identitat i adreça del responsable del tractament del fitxer de DCP.

A part d'aquests dos articles, la resta d'articles que parlen sobre els principis de protecció s'han tingut tots en compte (excepte articles sobre dades especialment protegides, que no és el cas).

DRETS DE LES PERSONES

Pel que respecten els drets de les persones, jo sóc tant el responsable del fitxer com l'encarregat del tractament, així doncs no cal complir cap article d'accés a dades en compte de tercers, però si que es permet als afectats (usuaris de l'aplicació) poder preguntar-me i sol·licitar qualsevol cosa sobre les seves dades emmagatzemades. Per això s'usa *phpMyAdmin* i s'ofereix la direcció del responsable del tractament als usuaris, per poder respondre a preguntes i sol·licituds d'aquest tipus de manera ràpida.

NIVELL DE SEGURETAR DE LA BASE DE DADES

Al fitxer amb les DCP està pensat aplicar-li un **nivell de seguretat alt**. Això és degut a que tot i que dels usuaris només es guarden nom i dades de contacte, es guarden els missatges d'informació enviats entre usuaris, que tot i que s'encripten, poden contenir informació d'ideologies i altres dades especialment protegides.

Al tenir un nivell de seguretat alt s'han realitzat les següents accions per a l'aplicació web:

- Identificació i autentificació personalitzada.
- Procediment d'assignació i distribució de contrasenyes.
- Les contrasenyes es guarden de manera xifrada.
- Límit d'intents reiterats d'accés no autoritzat a tres.
- La transmissió de missatges a través de la xarxa és xifrada amb els protocols **TLS** i **DTLS**.
- Les dades referents als usuaris es guarden de manera xifrada en els suports.

A part d'aquests requisits, el nivell de seguretat alt requereix moltes més funcionalitats, com per exemple l'obligació de fer auditories cada 2 anys, fer copies de seguretat, tenir un registre dels accessos a la BD, control d'accés físic al local on hi ha el servidor... Tots aquests requisits aliens a la programació no s'han tingut en comte; respecte el desenvolupament i programació l'únic requisit que s'incompleix és l'obligació de demanar el canvi de contrasenya cada any.

Per enviar les **dades xifrades a través de la xarxa** de manera encriptada s'han usat els protocols de xarxa a nivell d'aplicació de WebRTC, Socket.IO i HTTPS. En tots aquests protocols de la capa d'aplicació s'usa en capes inferiors els protocols TLS i DTLS com s'ha explicat en l'apartat de les tecnologies usades.

Per altre banda, per guardar les dades referents als usuaris de manera encriptada s'usa la funció hash del mòdul bcrypt i les funcions *aes_decrypt* i *aes_encrypt* [34].

10. Implementació i proves

10.1. Procés de desenvolupament

Per implementar el sistema, els quatre grans passos que es van seguir varen ser:

1- Implementar un sistema de videoconferència de manera local (tot el codi resideix en la part client i el servidor només s'encarrega de proporcionar la pàgina Web).

2- Dissenyar i implementar el sistema de videoconferència de manera distribuïda (el servidor a part de proporcionar la pàgina també s'encarregava del protocol de senyalització).

3- Dissenyar i implementar la part de gestió d'usuaris.

4- Relacionar-la la part de gestió d'usuaris amb la de videoconferències.

Seguidament es mostra per cada part els principals problemes i solucions obtingudes:

1- Implementació d'un sistema de videoconferència de manera local

Per la implementació d'aquesta primera part es van seguir els passos del *llibre* ***Real-Time Communication with WebRTC*** [35] i l'exemple d'implementació [36].

Els principals problemes els vaig tenir en *Chrome,* ja que perquè funcionés el sistema va ser necessari passar d'usar el protocol HTTP per a l'obtenció de pàgines a usar HTTPS.

Per passar a usar HTTPS cal instal·lar certificats SSL i canviar la configuració els arxius *default-ssl.conf* i *000-default-ssl.conf*.

000-default-ssl.conf conté les característiques referents al servidor *Apache* no segur ubicat sempre al port 80. Si es volen redirigir les peticions HTTP a *Apache* segur és necessari introduir aquesta línia en el fitxer:

```
Redirect permanent / https://kidd.udg.edu
```

La línia redirigeix totes les peticions fetes al port 80 al port 433 on hi el servidor d'Apache segur

default-ssl.conf conté la configuració del servidor *Apache* segur ubicat al port 443. En aquesta configuració, per poder usar el protocol HTTPS, cal afegir la ruta a on es troben els certificats SSL descarregats.

```
SSLEngine on
SSLCertificateFile      /etc/letsencrypt/live/kidd.udg.edu/fullchain.pem
SSLCertificateKeyFile   /etc/letsencrypt/live/kidd.udg.edu/privkey.pem
SSLCertificateChainFile /etc/letsencrypt/live/kidd.udg.edu/fullchain.pem
```

Antiguament els certificats SSL eren de pagament però actualment Let's Encript proporciona certificats per HTTPS de manera gratuïta.

La instal·lació d'un certificat SSL a través de LET'S ENCRIPT per Ubuntu està especificada a l'enllaç [37] i la manera d'incorporar els nous certificats a la configuració del port 443 d'Apache està en l'enllaç [38] de l'apartat de bibliografia.

A banda d'això la programació d'aquest apartat no m'ha portar més problemes.

2- Sistema de videoconferència distribuït

Al final de la implementació d'aquesta segona part s'obté l'API ASWRTC i la part de la API ASSIO referent al *namespace* ***grup***.

Per a poder passar a fer videoconferències entre usuaris de diferents *hotst* és necessari afegir un protocol de senyalització al codi obtingut en l'apartat 1 per poder intercanviar els objectes SDP entre usuaris. Per implementar el protocol de senyalització bidireccional entre client i servidor s'ha usat Socket.IO.

El protocol de senyalització que s'ha dissenyat i implementat és el següent:

Tenint en compte que tenim una videoconferència el la que hi ha l'usuari 1 i entra l'usuari 2 els missatges que s'envien es mostren en el diagrama d'interacció de la figura 24:

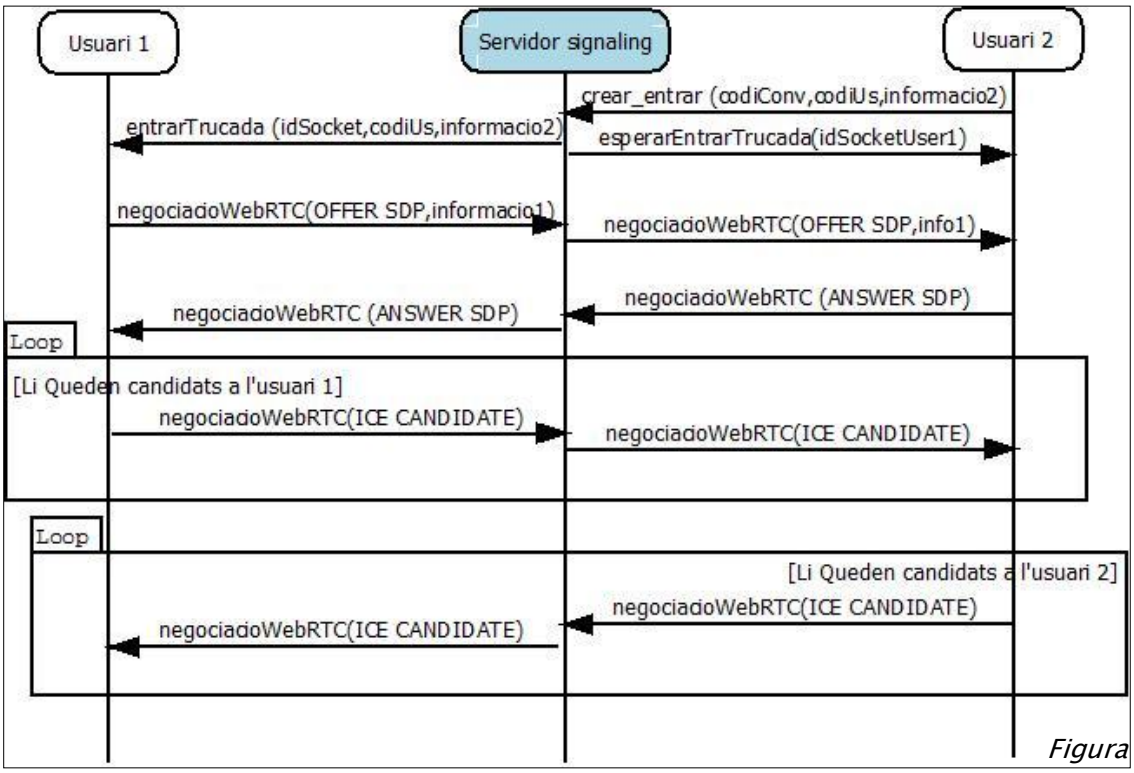


Figura 24

Encara que es mostri de manera seqüencial el procés d'ICE Candidates es fa paral·lel a al intercanvi de SDP ja que en els navegadors actuals s'usa el Trickle ICE Agent.

Tots els missatges referents al intercanvi de dades *WebRTC* van dintre dels missatges *negociacioWebRTC* i l'únic que fa el servidor és reenviar-los. Els demés missatges són per a avisar quan un usuari ha entrat o sortit a la videoconferència.

Els tipus de missatges que enviem a través de *negociacioWebRTC* són:

-OFFER :

Conté la descripció del SDP local del *peer* que comença el protocol de senyalització. Amb l'enviament d'un missatge d'aquest tipus s'espera enviar a l'altre peer el seu SDP local, i

quan l'altre peer hagi modificat la descripció remota amb el SDP rebut, respondrà un missatge de tipus ANSWER amb el seu SDP local.

-**ANSWER** :

Conté la descripció del SDP local de l'usuari que ha rebut un OFFER. La finalitat d'aquest missatge és informar a l'altre *peer* de la seva descripció de sessió.

-**CANDIDATE**

Conté un parell @IP i port pels quals es podria enviar i rebre fluxos multimèdia usant el protocol SRTP. Quan l'altre peer rep l'ICE Candidate ho notifica a l'ICE Agent per que s'encarregui de comprovar si es poden enviar dades cap aquella @IP i port indicats.

-**BANDWITHRENEGOTIATION**

Conté l'SDP local amb les noves VDT d'un usuari que ha detectat que un dels tracks rebuts del *PeerConnection* està enviant més bits per segons dels que la xarxa li permet. Un missatge d'aquest tipus permet modificar la VDT en temps real de tot un objecte *MediaStream*. L'usuari que rep un missatge d'aquest tipus per una *PeerConnection* actualitza l'SDP remot del *PeerConnection* per a passar a utilitzar les noves VDT que l'SDP marca.

Per implementar la part de senyalització de la videoconferència s'han utilitzat la API Socket.IO del *namespace* ***grup*** especificada en l'apartat 8.2.5 de disseny i s'ha seguit el diagrama d'interacció dibuixat a sobre.

Un cop establertes comunicacions d'àudio i vídeo bàsiques es van afegir totes les **funcionalitats de control de fluxos locals i remots**. Per notificar als demés usuaris de la conversa de que hem canviat l'estat d'un flux local o remot s'envia un una missatge del tipus adient través del canal DataChannel dels PeerConnections. Això permet no carregar el servidor i aprofitar que tenim una connexió P2P que ens permet enviar dades més ràpidament.

Pel que fa la implementació de la **gestió de fluxos locals es duen a terme les següents accions** (al final de cada acció es notifica als demes usuaris de la conversa que es veuen afectats a través del *DataChannel*):

Parar vídeo local:

S'esborra el *MediaStreamTrack* de tots els *PeerConnection* que connecten amb els usuaris remots, d'aquesta manera es deixa d'enviar la senyal de vídeo i no es consumeix velocitat de transmissió (passa a ser 0 pel medi de vídeo). Per tal de que un cop esborrat el *MediaStreamTrack* no s'hagi de tornar a fer la renegociació de SDP's es guarda l'objecte *MediaStreamTrack* tret del *PeerConnection* per a que, quan es torni a engegar es pugui re aprofitar el *track* i l'*ICE Candidate* que s'usava.

Engegar vídeo local:

Podem tenir 2 caos, si anteriorment s'havia parat el vídeo i tenim el *MediaStreamTrack* guardat només cal tornar afegir el Track a la llista de *streams* que s'envien de cada PeerConnection. Altrament, si l'usuari havia entrat sense vídeo i es el primer cop que l'engega cal fer crear un nou *PeerConnection* (amb cada usuari remot) per poder enviar el *MediaStreamTrack* creat. Aquest segon cas és degut a que si l'usuari no tenia vídeo l'objecte

MediaStream que es crea al inici no té la referència al dispositiu que capta el vídeo i cal crear un nou *MediaStream* que agafi les dades de la càmera.

Parar i engegar àudio local:

Es fa igual que pels *MediaStreamTrack* de vídeo, però en aquest cas el *MediaStreamTrack* que es guarda o es crea es de tipus d'àudio.

Per la gestió de fluxos remots s'usen diferents accions:

Parar vídeo remot:

Quan es para el vídeo d'un usuari remot, per a no mostrar-lo, el que es fa és canviar l'estat del *MediaStreamTrack* de vídeo que es rep de ***enabled*** a ***disabled***. Al fer això, a diferència d'abans no es talla l'enviament de flux a través del *PeerConnection* ja que això només pot fer-ho l'usuari que envia el *MediaStream*, el que es fa en aquest cas es alliberar el processament del vídeo a la GPU ja que el flux de vídeo es rep però no es mostra.

Engegar vídeo remot:

Canvia l'estat del *MediaStreamTrack* del *PeerConnection* remot de ***disabled*** a ***enabled***. Al fer això la GPU torna a processar el flux de vídeo rebut i el mostra per pantalla.

Parar i engegar àudio remot:

Es fa igual que pels *MediaStreamTrack* de vídeo, però en aquest cas el *MediaStreamTrack* que s'activa o es desactiva és d'àudio.

Com s'ha dit, per enviar tot tipus d'esdeveniments entre usuaris d'una conversa s'utilitza un DataChannel. El canal de dades està incrustat a cada *PeerConnection* entre dos usuaris i permet enviar "Strings" entre els dos usuaris del peer. Aquest canal inicialment no es tenia previst establir-lo i es volia usar Socket.IO per l'intercanvi d'esdeveniments, però per tal d'independitzar el màxim el sistema de videoconferències del servidor, al final es va optar per intercanviar dades entre usuaris de manera directe (a través del *PeerConnection*).

La resta de funcionalitats de l'API ASWRTC (gravació, obtenció d'estadístiques i limitació d'ample de banda) s'han explicat en l'apartat 7, ja que per aquestes funcionalitats s'usen llibreries per a simplificar la feina.

D'aquestes funcionalitats, es van tenir problemes amb la llibreria ***getStats,*** ja que no funcionava la interfície de la llibreria i es van haver d'accedir a mètodes més interns i fer un preprocés de les dades obtingudes semblant al que feia la llibreria de manera no actualitzada.

La implementació de la funcionalitat de la gestió dinàmica de les VDT també va estar bastant complexa. Per afegir la funcionalitat de detecció i solució de canals inestables va ser necessari conèixer bé el protocol SDP i comprovar que un objecte SDP es pot modificar dinàmicament un cop la *PeerConnection* està establerta entre dos usuaris.

3- Gestió d'usuaris

Un cop es tenia la part de videoconferències es va implementar tota la part referent a la interfície pels usuaris, tant la part *front-end* com la de *back-end*. Per la part *back-end* es va implementar la base de dades i les API's REST i Socket.IO pel *namespace* **usuari**. Per la part *front-end* es varen programar tant interfícies com la lògica d'Angular.JS i JQuery.

Els principals problemes que he tingut en la programació d'aquesta part es mostra en la següent llista de (problema - solució):

Incorporar els certificats LET'S ENCRYPT al servidor NodeJS.

PROBLEMA

La API REST programada en *NodeJS* usava el protocol HTTP per a rebre i enviar dades i pel compliment de la llei LOPD era necessari usar HTTPS.

SOLUCIÓ

Aprofitar certificats LET'S ENCRYPT que s'usen al servidor *Apache*. Com que els certificats ja estaven descarregats només cal indicar-li al servidor *Node* que els utilitzi per així poder treballar amb el protocol HTTPS. Per importar els certificats cal escriure el següent:

```
var credentials = {
  key: fs.readFileSync('/etc/letsencrypt/live/kidd.udg.edu/privkey.pem'),
  cert: fs.readFileSync('/etc/letsencrypt/live/kidd.udg.edu/cert.pem')
};
var app = express();
//Abans de crear el servidor es defineixen certes característiques sobre les peticions HTTPS
require('https').createServer(credentials,app).listen(8080);
```

Problemes amb les peticions a la API REST.

PROBLEMA

No es podien fer crides de tipus PUT, POST o DELETE a la API REST des de adreces IP amb dominis diferents al del servidor.

SOLUCIÓ

Usant el mòdul *express* es varen poden definir certes característiques sobre els peticions HTTPS, en aquest cas vam indicar que es permetessin les accions PUT, POST o DELETE des de dominis diferents seguint l'exemple del vídeo [39]:

```
//Configuració per poder fer peticions cross-domain
app.use(function(req, res, next) {
  res.header('Access-Control-Allow-Origin' , '*');
  res.header('Access-Control-Allow-Methods' , 'GET,PUT,POST,DELETE');
  res.header('Access-Control-Allow-Headers' , 'content-type');
  next();
});
app.use(bodyParser.urlencoded({extended: true}));
app.use(bodyParser.json());
//Es crea el servidor NodeJS
require('https').createServer(credentials,app).listen(8080);
```

Gestió de processos concurrents en NodeJS.

PROBLEMA

Per a respondre peticions de la API REST i Socket.IO alguns cops era necessari esperar a que acabessin ***n*** peticions a la base de dades. El fet de que *Node* treballi de manera concurrent feia que al enviar ***n*** peticions en el mateix moment no es poguessin recuperar els resultats que aquestes retornaven degut a que es perdia el fil d'execució.

SOLUCIÓ

Utilitzar el mòdul **ASYNC** per a poder sincronitzar els **n** processos asíncrons. En aquest cas s'han usat els mètodes ***async.each()*** , ***async.waterfall()*** i ***async.forEachOf*** .

Exemple realitzar n peticions iterant un vector d'elements:

```
var resultats = [];
async.each(vectorAlterar, function(elementVector, lastCallback) {
  //Petició x, on x ∈ {0..vectorAlterar.length-1)}
    async.waterfall([
      function(callback){
        //Petició concurrent amb elementVector
        resultat.push(resultat petició);
        callback();
      }
    ],function (res){//Quan acaba la petició feta dins la funció de waterfall s'entra aquí
      lastCallback();
    });
  }, function(err) { //Funció quan han acabat totes les peticions concurrents de vectorAlterar
      //Acció un cop acabades les peticions concurrents, els resultats estan a resultats
    }
});
```

Exemple realitzar n peticions iterant un vector d'objectes:

```
var results = [];
async.forEachOf(vectorobjectesAlterar, function (valorObkjecte, clauObjecte, lastCallback) {
  //Petició x, on x ∈ {0.. vectorobjectesAlterar.length-1)}
    async.waterfall([
      function(callback){
        //Petició concurrent amb valorObkjecte i clauObjecte
        results.push(resultat petició);
        callback();
      }
    ],function (res){ //Quan es crida callback  s'entra a aquesta funció
      lastCallback(); //Es crida a la funció que unirà tots els resultats de les
                      vectorobjectesAlterar.length peticions asíncrones.
    });
  }, function(err) {//Acció un cop acabades les peticions concurrents, els resultats estan a results }
});
```

Per a la utilització d'aquests mètodes de la llibreria ASYNC s'han consultat els enllaços [40], [41], [42] i [43].

Unir els controladors per a *moduls* amb diferents vistes.

PROBLEMA

Es va separar la interfície de la pàgina de gestió de l'usuari en varies vistes, associant un controlador diferent a cada vista. Però hi havien certes dades generals que s'obtenien de la API REST que eren necessàries per tots els controladors.

SOLUCIÓ

Per no realitzar la mateixa petició a la API REST en cada controlador, es va usar el patró *Factory* d'angular per augmentar la cohesió entre controladors i poder compartir les dades entre ells. D'aquesta manera només cal fer una petició HTTPS i si un controlador actualitza dades de l'objecte creat per la Factory s'actualitzen per a tots els controladors.

Xifrat del contingut de la BD.

PROBLEMA

Per a complir amb la LOPD era necessari xifrar tant el contingut de la BD referent a les dades personals dels usuaris com xifrar els missatges que viatgen a través de la xarxa.

SOLUCIÓ

Es va utilitzar el mòdul BCRYPT per a xifrar la contrasenya amb una funció hash i els mètodes AES_ENCRYPT i AES_DECRYPT.

Actualització de les vistes quan es criden els controladors des del *back-end*.

PROBLEMA

Quan es cridava un mètode del controlador per actualitzar la seva vista, si el mètode es cridava per la generació d'un esdeveniment provinent d'un lloc diferent a la vista, al actualitzar les dades del controlador la vista no s'actualitzava.

SOLUCIÓ

El problema està relacionat amb les variables $watch, $diges i $apply de l'objecte $scope d'AngularJS. En JavaScript tradicional el navegador informa al JavasScript dels canvis a la interfície a través de ***l'event loop***.

AngularJS estén la funcionalitat de l'*event loop* a través de l'***Angular Context,*** que captura els esdeveniments que després ens permeten actualitzar la pàgina automàticament. Quan es calcula un esdeveniment en angular es criden els següents mètodes de la l'objecte $scope:

$apply -> $digest -> wathcers

Cada variable que es crea en angular se l'hi associa un watcher que avalua constantment si la variable ha canviat el seu valor.

Cada cop que s'actualitza una variable angular escrida el mètode $apply, que crida a $digest. Llavors $digest itera tota la llista de wathers associats a cada variable per mirar si ha canviat algun valor, si el valor canvia llavors $digest actualitza la vista.

El problema es que el mètode *$apply* només es crida si hi han canvis dins de l'scope de la vista *Angular* (que és on actua l'Angular context). D'aquesta manera si rebem una resposta

de la API REST o Socket.IO, que són esdeveniments que l'Angular context no avalua, llavors no es crida el mètode *$apply* encarregat de notificar a *$digest* de que hi ha un nou esdeveniment que pot produir canvis.

Per solucionar això el que s'ha de fer cridar manualment (des de dins el controlador) el mètode *$apply* passant-li una funció, i dintre d'aquesta funció s'han d'actualitzar les variables que volem que canviïn a la vista.

```
$scope.$apply(function(){
    $scope.variable = nouValor ;
});
```

D'aquesta manera es com actualitzem la vista quan es crida el controlador des d'esdeveniments que l'Angular context no cobreix.

4- Unió de la part 2 i 3

Quan des del perfil de l'usuari es realitza una trucada i s'accepta s'obra una nova pàgina en la que es du a terme la videoconferència entre els usuaris d'una conversa. Per a passar les dades entre les dos pàgines s'utilitza el un formulari PHP amb els següents camps:

```
nomUsuari = "Nom de l'usuari que entra a la videoconferència"
idUsuari = "Identificador únic de l'usuari que entra a la videoconferència "
avatar_url = "URL de l'imatge que es vol mostrar junt amb el nom de la videoconferència"
nomConversa = "Nom de la videoconferència"
idConversa = "Identificador de la conversa i nom del grup de la videoconferència"
vídeo = "Cert si l'usuari vol emetre senyal de vídeo a través de la càmera"
àudio = "Cert si l'usuari vol emetre senyal de àudio a través del micròfon"
gravació = "Cert si l'usuari permet la gravació dels fluxos multimèdia locals i remots"
amplesbanda = "Vector amb els amples de banda de pujada i baixada pels medis d'àudio i vídeo"
```

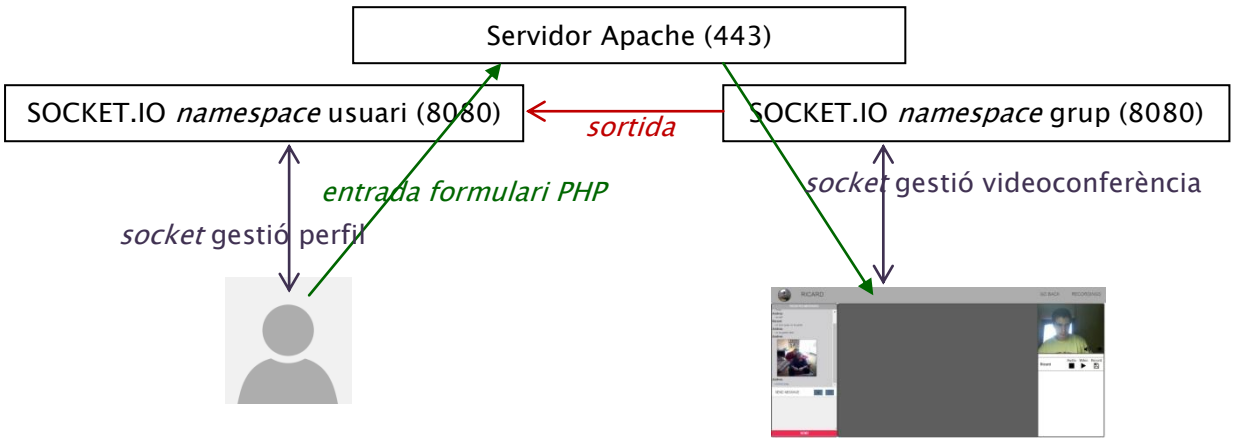
Gràcies a aquest formulari que s'envia al servidor per obrir la pàgina de videoconferències *https://kidd.udg.edu/c_videoconferencia.php* es permet que qualsevol sistema pugui incorporar la part de videoconferència a la seva pàgina Web només fent una crida al nostre servidor i passant-li el formulari emplenat. O si ho prefereix, es pot implementar el ***namespace*** **grup** a un servidor *NodeJS* proporcionat i tot funcionarà correctament.

L'única restricció que hi ha en el formulari de videoconferència es que no pot haver-hi cap *nomConversa* igual en un moment determinat.

Tornant al nostre sistema, s'usa l'identificador únic de la conversa per a crear sales de videoconferències úniques; cada conversa només pot crear una videoconferència. Per fer això un cop s'ha iniciat una videoconferència es bloqueja la possibilitat de realitzar més trucades fins que l'últim usuari surt de la videoconferència.

Un cop l'últim usuari surt es permet tornar a trucar; En aquest cas per informar a la part de gestió de perfil que un usuari ha sortit de la videoconferència s'usa el protocol Socket.IO enviant un missatge de fi de trucada al *socket* actiu que té l'usuari per a la gestió de perfil.

La implementació de la unió entre la videoconferència i gestió de perfil es pot representar amb el següent esquema:



En el dibuix es representa que cada usuari té un *socket* sempre actiu per a la gestió del perfil que s'usa per a anar actualitzant les dades en temps real, i per cada videoconferència que té activa té un nou *socket* que s'usa pel protocol de senyalització i control d'usuaris que hi ha a la videoconferència.

Quan l'usuari entra a una videoconferència demana la pàgina al servidor, amb el formulari PHP emplenat, i aquest el respon amb l'obertura de la pàgina de videoconferències en una nova finestra. Allà comença el protocol de senyalització implementat en la segona part i s'inicien els intercanvis de dades i fluxos multimèdia.

Al sortir de la videoconferència, com que l'usuari segueix tenint el *socket* de gestió de perfil obert se li notifica que l'usuari ha tancat la pàgina de videoconferències, informant del nombre d'usuaris que encara són a dintre. En el cas de ser l'últim de sortir el gestor de perfil acaba amb la conversa i permet trucar de nou, altrament s'activa el botó que permet a l'usuari tornar entrar a la videoconferència que encara està activa.

Quan l'usuari surt de la conferència tancant la finestra el servidor rep un esdeveniment ***disconnect*** per el *socket* amb el qual s'ha perdut la connexió. En aquest moment es quan es notifica tant a la part de gestió de l'usuari com a la resta d'usuaris que encara estan a la videoconferència que l'usuari ha sortit de la trucada.

10.2. Proves realitzades i resultats

El sistema s'ha anat testejant a mesura s'han anat acabant les quatre parts d'implementació que s'han exposat en l'anterior apartat. Llavors, quan ja tot estava unificat s'han realitzat 5 escenaris diferents amb diferents característiques de xarxa per a comprovar que el sistema de videoconferències limitava l'ample de banda correctament i que els amples de banda escollits pels usuaris eren els correctes.

Finalment, per comprovar que la API ASWRTC detectava i solucionava problemes d'inestabilitats en l'intercanvi de fluxos d'una conversa (degut a una mala inicialització de les VDT usades) s'ha utilitzat només un escenari amb varies variables per veure com funcionava durant el transcurs d'una videoconferència.

Seguidament s'exposen les proves realitzades per a la gestió de la interfície dels usuaris i posteriorment es mostraran els 5 escenaris construïts i com han respost en diferents circumstàncies. Finalment es mostra el test realitzat sobre la gestió de VDT a la API ASWRTC.

Els testos per a les funcionalitats en que l'usuari i intervé estan separats segons les diferents planes del sistema.

PLANA DE REGISTRE I AUTENTIFICACIÓ

En aquest apartat s'ha testejat la part de registre i autentificació. Les coses comprovades han estat:

- Registrar 10 usuaris diferents.
- Comprovar que no es pot registrar un usuari amb un nom i/o correu ja existents.
- Comprovar que un usuari si falla tres cops en l'autentificació no se li deixa accedir a la pàgina.
- Assegurar que la contrasenya es guarda de manera encriptada.
- Comprovar que els usuaris no autentificats no poden accedir a l'aplicació.

PLANA DE GESTIÓ PERSONALITZADA

S'han testejat tots els casos d'ús referents a l'actor usuari identificat i s'ha comprovat també el funcionament dels events de negoci que llança l'actor sistema:

USUARI IDENTIFICAT

- S'han comprovat els diferents filtres per a reduir les llistes d'usuaris, amics i peticions.
- Comprovar que al enviar una petició d'amistat la petició es mostra degudament.
- Comprovar totes les situacions de resposta de peticions (tant binaries com múltiples). S'ha mirat que funciones correctament les següents situacions:
    - Que quan un usuari rebutja una petició aquesta s'esborri en temps real per la resta d'usuaris afectats.
    - Quan un usuari accepta una petició canvi d'estat i no es permeti respondre-la de nou.
    - Quan tots els usuaris han acceptat la petició passi a crear-se la conversa i a mostrar-se en la interfície l'opció d'accedir-hi.
- Creació de converses múltiples (s'han creat converses de 3, 4 i 5 usuaris a partir de la llista d'amics).
- Diferents actualitzacions de perfil (canvi de foto, nickName i correu).
- Control amb la selecció de converses (seleccionar i desseleccionar tot tipus de conversa)

- Enviament de tot tipus de missatge de text (caràcters especials), arxius i imatges de diferents formats i comprovar que es xifren i es desxifren correctament alhora de guarda'ls a la BD.

- Ús de totes les possibles característiques de videoconferència (vídeo, àudio i/o gravació).

- Comprovació de la persistència de dades per a tots els casos anteriors fent entrar usuaris als que se'ls hi ha enviat peticions i missatges quan no estaven connectats.

SISTEMA

- Realització de càlculs de velocitat de transmissió en xarxes amb diferents capacitats i durada del test. Les durades segons la capacitat de les xarxes són les següents:

  - Xarxa de molt baixa qualitat (VDT baixada: 1136kbits VDT pujada: 264kbits):

    Temps test de la VDT de pujada: 1 minut 6 segons

    Temps test VDT de baixada: 4 minuts 94 segons

  - Xarxa de baixa qualitat (VDT baixada: 9216.56kbits VDT pujada: 788.48kbits):

    Temps test de la VDT de pujada: 8.49 segons

    Temps test VDT de baixada: 1,655 segons

  - Xarxa de mitja qualitat (VDT baixada: 2538.63kBytes VDT pujada: 772.627kBytes):

    Temps test de la VDT de pujada: 4 segons

    Temps test VDT de baixada: 12,68 segons

  - Xarxa alta qualitat (VDT baixada: 7812.192kBytes VDT pujada: 9946.68kBytes):

    Temps test de la VDT de pujada: 1,25 segons

    Temps test VDT de baixada: 0,98 segons

  Els amples de banda de les xarxes que el test ha assignat s'han comprovat amb diferents testos de velocitat i els resultats han estat molt semblants.

- Comprovació del sistema de geocoalització. En aquest cas, a diferència del test de VDT, la geocolitació que ofereix la API de geocoalització per satèl·lit no es tant precisa i alguns cops les coordenades que dona sobre la ubicació de l'usuari poden variar fins a 30 km. Durant varies proves sobre un ordinador de taula ubicat a Santa Coloma de Farners alguns cops, la ubicació que ha donat l'API han estat punts ubicats a Girona, la resta de cops ha donat la posició correcte com era d'esperar.

PLANA DE VIDEOCNFERÈNCIA

S'han testejat tant el xat no persistent com tota la part de gestió de fluxos locals i remots amb multi conferències de fins a QUATRE usuaris.

Després de provar totes les funcionalitats en videoconferències entre 2, 3 i 4 usuaris tot funciona correctament excepte l'enviament de fitxers en el navegador *Firefox* i la gravació de només vídeo en *Chrome*. En el cas de l'enviament d'arxius en Firefox l'arxiu s'envia correctament, però no s'ha pogut crear una URL blob que permeti la descarrega del fitxer tal com es fa amb *Chrome*, això fa

que el problema no estigui en l'enviament sinó en la creació del fitxer a partir de les dades rebudes.

En el cas de la gravació únicament de vídeo en *Chrome* el problema és menor, ja que la gravació de vídeo sense àudio es poc corrent. El problema en aquest cas és degut a la llibreria que s'usa per la gravació, tot i que probablement en la següent actualització el creador de la llibreria el solucioni.

Referent a la llibreria sobre WebRTC, l'obtenció d'estadístiques s'ha provat en varis navegadors Chrome de diferents versions i només funciona en les últimes versions de Chrome i Firefox. De fet la llibreria funcionava en la versió 51 de Chrome però es va preferir modificar-la perquè funciones en la versió 52 tot i perdre el suport per a Chrome 51.

Explicades les proves habituals realitzades per les diferents funcionalitats implicades en la interfície de l'usuari la major part de testeig s'ha dedicat a la comprovació de que la regulació de la VDT en les videoconferències que fa la api ASWRTC funcionés correctament.

Per comprovar que la regulació de la VDT funcionava correctament s'han muntat 5 escenaris diferents en que els usuaris participants tenien xarxes amb VDT's molt diferents. Llavors, **per tastejar que ASWRTC podia canviar les VDT usades durant el transcurs d'una sessió s'ha realitzat un únic escenari amb varies proves diferents.**

ESCENARI 1

En aquest escenari es pretén provar que el servidor STUN funciona correctament i que es regulen les VDT segons el resultat del test de velocitat que se li passa com a paràmetres a ASWRTC .

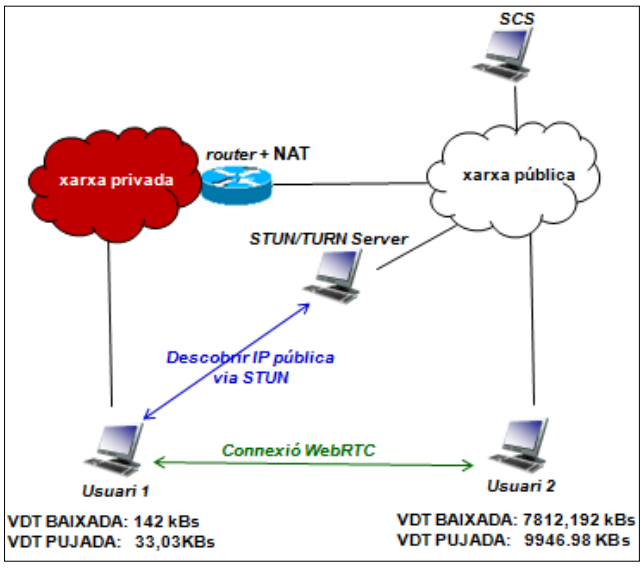


Figura 25

En l'escenari, tal com es mostra en la figura 25, l'usuari 2 està en una xarxa simètrica en que la @IP és pública, per això, encara que alhora d'obtenir els *ICE candidates* obtingui candidats de tipus STUN i TURN no li faran falta, degut a que la @IP del *host* ja es visible per qualsevol altre usuari. Per altre banda l'usuari 1 està en una xarxa privada i en aquest cas és necessari que la @IP que agafa sigui la adreça publica del router NAT que li ha de proporcionar el servidor STUN.

Com que només un dels 2 usuaris està en una xarxa privada, l'usuari de la xarxa privada pot iniciar l'intercanvi de dades per a cada flux multimèdia, fent que així al enviar ell primer els

missatges se li quedin obertes les entrades a la taula NAT per cada port que usa. D'aquesta manera, quan l'altre usuari respon se li permetrà enviar dades als ports que se li han obert i la comunicació entre els dos usuaris serà directe entre navegadors (P2P).

En aquest cas les VDT escollides pel sistema han estat les següents:

Usuari 1:

VDT baixada = 142kBs / 100 * 60 = 681.6 kbs

VDT pujada = 33,03kBs / 100 * 70 = 184.4 kbs

A partir de la llista estàtica de VDT segons el medi i còdec s'han escollit les següents VDT per a la videoconferència:

VDT baixada àudio= 40 kbs VDT baixada vídeo = 600 kbs

VDT pujada àudio = 20 kbs VDT pujada vídeo = 128 kbs

Usuari 2:

VDT baixada = 7812,192kBs / 100 * 60 = 37498.6 kbs

VDT pujada = 9946,98kBs / 100 * 70 = 55703.6 kbs

A partir de la llista estàtica de VDT segons el medi i còdec que s'ha construït, les VDT escollides per a la videoconferència són:

VDT baixada àudio= 256 kbs VDT baixada vídeo = 15360 kbs

VDT pujada àudio = 256 kbs VDT pujada vídeo = 15360 kbs

En aquest cas s'agafen les velocitats de transmissió de l'usuari 1. L'usuari 2 enviarà els fluxos multimèdia d'àudio i vídeo a 40 *kbs* i 600 *kbs* respectivament i rebrà una quantitat d'informació de 20kbs pel flux d'àudio i 128kbs de vídeo de l'usuari1.

Degut a la diferència de VDT entre el dos usuaris l'usuari 2 utilitza una qualitat de vídeo i àudio molt inferior a la que podria usar, però si no ho fa així i usa les seves velocitats de transmissió l'usuari 1 no podrà rebre totes les dades que l'usuari 2 li envia i començarà a perdre paquets. Per altre banda si l'usuari 1 vol enviar tantes dades com l'usuari 2 pot rebre s'està exigint que enviï més informació del compte i intenta enviar més dades de les que pot; en aquest cas l'usuari 2 també rep les dades saturades ja que s'acumulen els paquets a la central del proveïdor i es perden.

Amb les VDT establertes es va fer una videoconferència de 10 minuts i va funcionar correctament, però per tal de veure com responia el sistema si l'usuari 1 se li posaven unes VDT superiors a les de la seva xarxa es van canviar les VDT dels seus medis.

VDT baixada àudio= 40 kbs VDT baixada vídeo = 600 kbs

VDT pujada àudio = 32 kbs VDT pujada vídeo = 300 kbs

En aquest cas el flux d'àudio i vídeo enviats per l'usuari 1, l'usuari 2 els rebia amb constants talls i retard. Això és perquè amb les noves VDT l'usuari 1 li indica a l'usuari 2 que ell li enviarà 332 kb d'informació cada segon (sumant els medis d'àudio i vídeo) i l'usuari 2 els i exigeix. El problema però és que la xarxa en la que està l'usuari 1 només li permet enviar 263 kb d'informació per segon.

ESCENARI 2

En aquest escenari es prova la realització de videoconferències d'alta qualitat entre 2 usuaris situats en una xarxa pública. Per a fer aquest test s'han usat dos @IP públiques proporcionades pel grup de recerca BCDS. En aquest cas el servidor STUN no té cap utilitat ja que les @IP i ports usats per l'intercanvi de dades multimèdia dels dos hosts són visibles des de qualsevol put d'Internet, així doncs si no i hagués el servidor STUN+TURN funcionaria tot igual.

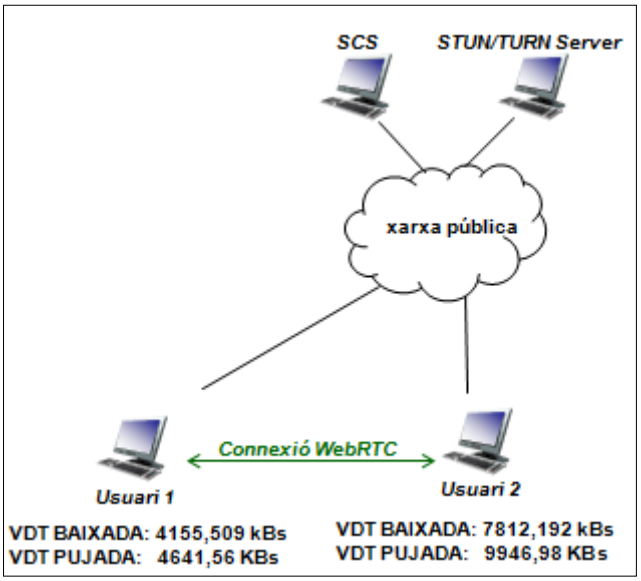


Figura 26

En l'escenari de la figura 26 les VDT escollides per a cada usuari són les següents:

Usuari 1:

VDT baixada = 4155,509kBs / 100 * 60 = 19946,444 kbs

VDT pujada = 4641,56kBs / 100 * 70 = 25992.736 kbs

A partir de la llista estàtica de VDT segons el medi i còdec que té el sistema, les VDT escollides per a la videoconferència de l'usuari 1 són:

VDT baixada àudio= 256 kbs VDT baixada vídeo = 15360 kbs

VDT pujada àudio = 256 kbs VDT pujada vídeo = 15360 kbs

Usuari 2:

VDT baixada = 7812,192kBs / 100 * 60 = 37498.6 kbs

VDT pujada = 9946,98kBs / 100 * 70 = 55703.6 kbs

Les VDT que el sistema a determinat (a partir de la qualitat de xarxa en al que està aquest segon usuari) són les següents:

VDT baixada àudio= 256 kbs VDT baixada vídeo = 15360 kbs

VDT pujada àudio = 256 kbs VDT pujada vídeo = 15360 kbs

Amb les velocitats usades el còdec VP8 permet l'intercanvi de vídeos full HD. I per tant la qualitat de les senyals tant de vídeo com d'àudio són molt millors que les del primer escenari, això és

perquè ja no existeix una limitació de les VDT per part d'un usuari; en aquest cas els dos usuaris estan en xarxes asimètriques i poden enviar-se una gran quantitat d'informació per cada segon.

Respecte aquest escenari inicial es va provar que passava si s'afegia un nou usuari amb unes VDT de xarxa semblant a les de l'usuari 2 a la videoconferència mantenint les mateixes VDT entre els usuaris.

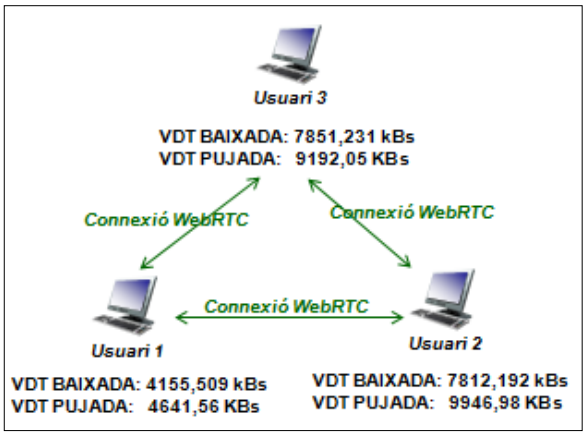


Figura 27

En aquest segon cas cada usuari ha de rebre i enviar dos streams d'àudio i dos de vídeo, això significa que la velocitat de transmissió és multiplica per 2 i per tant cal fraccionar en 2 la VDT de baixada i pujada que teníem disponible per a cada flux multimèdia. Si això no es fa, que es el que s'ha provat en aquest cas, els *bits per segon* enviats superen la VDT de la xarxa i es sobrecarrega. Això fa que els *streams* enviats comencin a acumular pèrdues i retard.

ESCENARI 3

En aquest escenari es confirma que per a dos usuaris que estan a la mateixa xarxa privada no és necessari l'ús del servidor STUN/TURN, en aquest cas la xarxa privada és asimètrica i de no molta qualitat. Respecte l'escenari 2, les VDT de pujada i baixada són molt menors i en la videoconferència s'observa una diferencia de qualitat molt notable (en especial en els *streams* de vídeo).

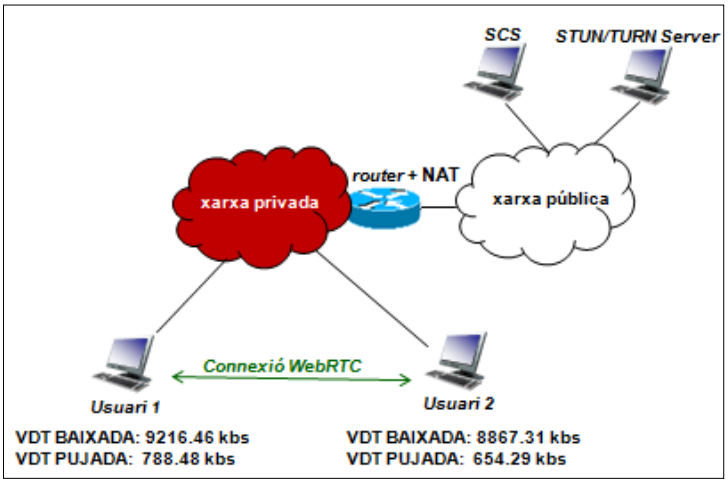


Figura 28

En aquest cas el sistema assigna les mateixes VDT pels dos usuaris tot i que en el test de velocitat els resultats siguin una mica diferents:

VDT baixada àudio= 180 kbs VDT baixada vídeo = 5120 kbs

VDT pujada àudio = 40 kbs VDT pujada vídeo = 400 kbs

Com que estan a la mateixa xarxa, l'ICE Agent de *WebRTC* escull com a candidats les adreces d'àmbit privat i no la pública del router. Els paquets de xarxa en aquest cas són recollits i reenviats pel ***Access Pont*** (ja que els dos estan en Wifi) i el canal de comunicació entre els dos navegadors no surt de la xarxa privada.

En aquest cas, la VDT de pujada escollida pels dos usuaris és la mateixa, 400kbs pel vídeo i 38kbs per l'àudio. Els valors escollits com a VDT de pujada són per a cada medi el valor menor entre la VDT local pel medi i la VDT de l'usuari remot pel mateix medi.

En aquest escenari s'han realitzat varies videoconferències d'entre un i deu minuts i només es van tenir problemes quan la xarxa presentava comportaments inesperats (proporcionava una VDT variant que al baixar molt el valor feia que la videoconferència es tornés inestable (per aquest fet es va proporcionar la funcionalitat de canviar les VDT durant la sessió)).

ESCENARI 4

En aquest escenari es posa a prova l'intercanvi de fluxos multimèdia entre xarxes privades diferents.

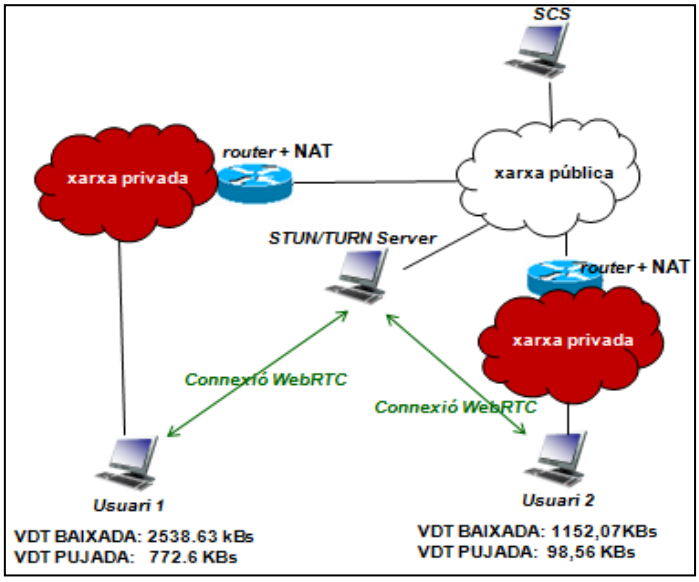


Figura 29

Com que la comunicació és entre 2 usuaris que estan en una xarxa privada, s'ha hagut de redirigir el canal *WebRTC* pel servidor TURN. Aquest fet no te perquè passar sempre, però si dos navegadors que estan en una xarxa privada no estaven comunicats a través d'un canal P2P moments previs a l'establiment del canal *WebRTC*, sempre es redirigirà el canal *WebRTC* pel servidor TURN.

Pel cas de la gestió de VDT és indiferent si el canal *WebRTC* necessita que el servidor TURN faci de pont, i en aquest cas les VDT que el sistema ha escollit per a cada usuari i medi abans d'iniciar la videoconferència han estat les següents:

Usuari 1:

VDT baixada = 2538,63kBs / 100 * 60 = 12185,424 kbs

VDT pujada = 772,6kBs / 100 * 70 = 4326,56 kbs

Segons el medi i còdec que s'ha construït les VDT escollides per a la videoconferència són:

VDT baixada àudio= 220 kbs VDT baixada vídeo = 10000 kbs

VDT pujada àudio = 150 kbs VDT pujada vídeo = 4000 kbs

Usuari 2:

VDT baixada = 1152,07kBs / 100 * 60 = 5529,936 kbs

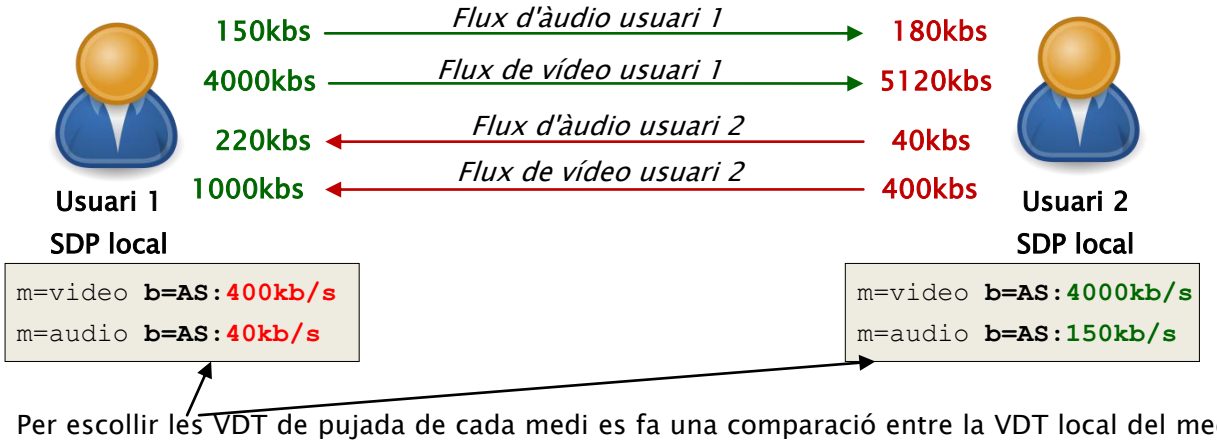
VDT pujada = 98,56kBs / 100 * 70 = 551,93 kbs

A partir de la llista estàtica de VDT segons el medi i còdec que s'ha construït les VDT escollides per a la videoconferència són:

VDT baixada àudio= 180 kbs VDT baixada vídeo = 5120 kbs

VDT pujada àudio = 40 kbs VDT pujada vídeo = 400 kbs

A partir de la VDT en les que pot enviar i rebre informació cada usuari es pot dibuixar els següent esquema de possibilitats:



Per escollir les VDT de pujada de cada medi es fa una comparació entre la VDT local del medi i la VDT remota pel mateix medi. En aquest cas, tal com s'indica en els objectes SDP, les VDT escollides són en per a tots els ***tracks*** la VDT local de pujada.

Amb aquestes velocitats de transmissió s'ha realitzat una videoconferència de 20 minuts i tot ha funcionat correctament.

Seguint amb el mateix escenari es va testejar que passava si es canviaven les VDT entre els dos usuaris obtinguen els dos següents objectes SDP:

Usuari 1 SDP local

```
m=video b=AS:4000kb/s
m=audio b=AS:150kb/s
```

Usuari 2 SDP local

```
m=video b=AS:400Kb/s
m=audio b=AS:40kb/s
```

En aquest cas l'stream de vídeo i àudio que envia l'usuari 2 no arriba correctament, ja que en l'objecte SDP de l'usuari 1 s'està indicant que l'usuari 2 ha d'enviar 4150kb/s d'informació sumant tots els *streams* multimèdia i l'usuari 2 pot enviar com a molt 551kb/s (tenint en compte que usa el 100% de la seva capacitat de xarxa).

A part de que els fluxos multimèdia de l'usuari 2 arribaven tallats i amb retard a l'usuari 1, el xat tampoc funcionava correctament. Durant la videoconferència s'han provat d'enviar

missatges i arxius, i els missatges enviats per l'usuari 2 arribaven amb molt retard a l'usuari 1 (degut a que els fluxos multimèdia sobrecarreguen la xarxa de pujada i no deixen espai per enviar res més).

Pel que respecta els fluxos multimèdia i missatges de xat enviats de l'usuari 1 a l'usuari 2 aquests funcionen correctament i l'usuari 2 rebia totes les dades amb normalitat; per tant l'embut està en la VDT de pujada de l'usuari 2.

ESCENARI 5

Aquest escenari es una extensió de l'anterior. El seu objectiu es comprovar com el sistema canvia les VDT escollides tenint en compte el numero d'usuaris que hi haurà a la videoconferència. Per a fer la prova s'ha realitzat una multiconferència entre 3 usuaris situats en xarxes privades diferents.

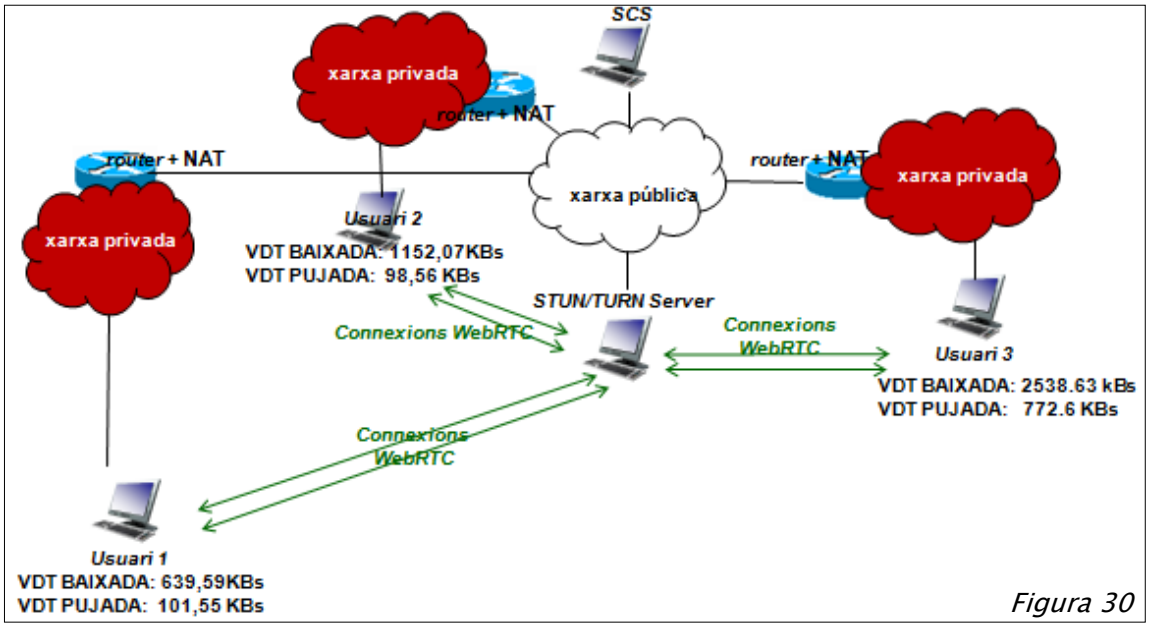


Figura 30

Al igual que en els anteriors escenaris, es mostra per cada usuari el resultat del test de velocitat i s'indica que tots els usuaris han de passar pel roter+NAT per poder interactuar amb hosts de la xarxa pública.

En aquest cas, totes les connexions entre els usuaris passen pel servidor TURN ja que tots estan en xarxes privades encobertes per NATS simètrics.

A partir dels resultats del test de velocitat que fa el sistema aquest cop es té en compte també que cada usuari ha d'interactuar amb 2 més fent així que les VDT obtingudes en el test es divideixin per dos. D'aquesta manera les VDT assignades a cada usuari per entrar a la videoconferència són les següents:

Usuari 1:

VDT baixada = 639,59kBs/100*60 = 3070.03kbs => 3070,03/2 = 1535,015kbs

VDT pujada = 101,55kBs/100*70 = 568,68kbs => 568,68/2 = 284,34kbs

Segons el medi i còdec que s'ha construït les VDT escollides per a la videoconferència són:

VDT baixada àudio= 96kbs VDT baixada vídeo = 1152kbs

VDT pujada àudio = 25kbs VDT pujada vídeo = 200kbs

Usuari 2:

VDT baixada = 1152,07kBs/100*60 = 5529.936kbs => 5529.936/2 = 2764kbs

VDT pujada = 98,56kBs/100*70 = 551.93kbs => 551.93/2 = 275,97kbs

A partir de la llista estàtica de VDT segons el medi i còdec que s'ha construït les VDT escollides per a la videoconferència són:

VDT baixada àudio= 128kbs VDT baixada vídeo = 2500kbs

VDT pujada àudio = 25kbs VDT pujada vídeo = 200kbs

Usuari 3:

VDT baixada = 2538,63kBs/100*60 = 12185,424kbs => 12185,424/2 = 6092,71kbs

VDT pujada = 772,6kBs/100*70 = 4326,56kbs => 4326,56/2 = 2163,28kbs

Segons el medi i còdec que s'ha construït les VDT escollides per a la videoconferència són:

VDT baixada àudio= 180kbs VDT baixada vídeo = 5120kbs

VDT pujada àudio = 100kbs VDT pujada vídeo = 2000kbs

En aquest cas cada dos usuaris es posen d'acord sobre les VDT que usaran a per enviar fluxos de dades a través del *PeerConnection* que els uneix. En aquest cas les VDT que el sistema ha escollit per cada medi de cada *PeerConnection* són les següents:

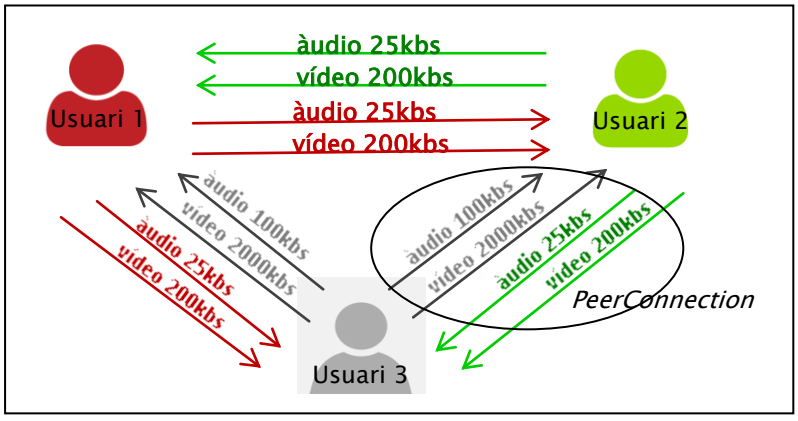


Figura 31

En les canals *WebRTC* cada usuari utilitza les mateixes VDT de pujada per a tots els *PeerConnection*, això és perquè en tots els casos la VDT de l'usuari local és menor a la VDT de baixada remota i els valors petits tenen preferència.

En aquest escenari s'ha realitzat una videoconferència de 25 minuts entre els 3 usuaris amb gravació, enviament de missatges, fitxers i imatges a través del xat, i gestió dels medis d'àudio i vídeo. Al haver tingut en compte que ara hi ha un usuari més (tenim tres *PeerConnection*) respecte l'escenari quatre, la videoconferència ha funcionat correctament.

CANVI DE LES VDT EN EL TRANSCURS D'UNA VIDEOCONFERÈNCIA

Per a tastejar la **capacitat de detectar quan en un *MediaStreamTrack* d'un *PeerConnection* està enviant més bps que els que la xarxa en la que es troben els dos usuaris li permet** es va fer un control de com evolucionaven les estadístiques obtingudes sobre els diferents *MediaStreamTrack* que participaven en una *PeerConnection* a partir de diferents situacions d'estrès. És a dir, per cada senyal multimèdia que rebia un usuari (àudio o vídeo) és va mirar quin valors estadístics se li assignaven tenint en compte diferents possibilitats.

Es van observar en especial els paràmetres de paquets rebuts, paquets perduts i limitació de resolució. Ja que eren els paràmetres que un cop consultada la API de ***getStats*** tenien més possibilitats d'acostar-nos a la detecció de converses inestables.

Per fer l'observació es va utilitzar l'**escenari quatre** assignant manualment les velocitats inicials de transmissió:

Recordem que en l'escenari 4 teníem dos usuaris amb les següents VDT de xarxa:

Usuari 1:

VDT baixada = 2538,63kBs kbs

VDT pujada = 772,6kBs kbs

Usuari 2:

VDT baixada = 1152,07kBs kbs

VDT pujada = 98,56kBs kbs

A partir d'aquí es van inicialitzar tres videoconferències amb les següents VDT per a cada usuari i es van obtenir els següents resultats, és important tenir en compte que la resolució màxima de vídeo estava limitada a 640x480 i si s'usen ***bitrates*** superiors a 1024 kbs no tenen efecte en la disminució de resolució :

Videoconferència 1:

Usuari 1: VDT pujada àudio inicial = 256kbs VDT pujada vídeo inicial = 15360kbs

Usuari 2: VDT pujada àudio inicial = 256kbs VDT pujada vídeo inicial = 15360kbs

Al començar la videoconferència tots els fluxos intercanviats tenien problemes, tots dos usuaris sentien l'àudio remot tallat i en comptes de 24 fps en els vídeos rebuts es mostraven 1 o 0 fps. En aquest cas no hi havia limitació de resolució en vídeo però el numero de paquets perduts en els *streams* de vídeo era d'un 80% respecte els esperats.

Videoconferència 2:

Usuari 1: VDT pujada àudio inicial = 150kbs VDT pujada vídeo inicial = 4000kbs

Usuari 2: VDT pujada àudio inicial = 150kbs VDT pujada vídeo inicial = 4000kbs

En aquest cas els fluxos d'àudio i vídeo rebuts per l'usuari 2 tenien poques inestabilitats i durant una videoconferència de 2 minuts no es detectaven grans quantitats de paquets perduts. En aquest cas es rebien una mitja de 442.7 paquets per segon respecte el flux de vídeo, i el percentatge de paquets perduts estava sobre un 5% del total de paquets esperats.

L'usuari 1 per altre banda rebia el vídeo i àudio molt tallats i la quantitat de paquets perduts era major a un 50%. L'usuari 2 enviava aproximadament 500 paquets per segon però només n'arribaven 100 a l'usuari 1.

Videoconferència 3:

Usuari 1: VDT pujada àudio inicial = 40kbs VDT pujada vídeo inicial = 400kbs

Usuari 2: VDT pujada àudio inicial = 40kbs VDT pujada vídeo inicial = 400kbs

Durant una videoconferència de 4 minuts no es va detectar pràcticament cap paquet perdut (un percentatge menor a un 1%) en els fluxos d'àudio i vídeo rebuts per part dels dos *peers*.

Després d'aquest resultat es va considerar que les variables *packetsLost* i *packetsReceived* podrien servir per a la detecció de *MediaStreamTracks* inestables.

Com a segona part del test, per a avaluar la **renegociació d'ample de banda** es va afegir la funcionalitat a l'API i es va veure com reaccionava en les tres videoconferències anteriors. En aquest cas els resultats, un cop assegurat que la implementació fos la correcte, van ser els esperats.

A l'API ASWRTC, un cop detectat un flux entrant inestable, passa per una escala de reduccions de VDT, on cada element és una possible suma de totes les VDT dels medis entrats en un *PeerConnection*. En el cas de detectar inestabilitats en el *PeerConnection* es redueix el valor actual al següent valor més petit de l'escala.

L'escala de VDT's actualment només conté tres possibles reduccions de VDT, que són { àudio: 96, vídeo: 1500},{ àudio: 32, vídeo: 200} i { àudio: 8, vídeo: 16}. Per exemple, si la VDT actual de baixada és superior a 1596 i els fluxos multimèdia rebuts són inestables es redueix la VDT a 96kbs i 1500kbs pels medis d'àudio i vídeo respectivament ja que es el primer valor de l'escala que la suma de les VDT dels seus medis no supera els kbs que actualment s'estan usant.

A partir d'aquí el resultat en les tres videoconferències ha estat el següent:

Videoconferència 1:

Aquest és l'únic cas en que els dos usuaris van acabar reduint les VDT inicials de 15616kbs a unes VDT obtingudes de l'escala de possibles VDT.

L'usuari 1, pocs segons després d'iniciar la conversa ja va reduir la VDT de baixada a 96kbs i 1500kbs pels medis d'àudio i vídeo respectivament, cosa que va implicar que l'usuari 2 no hagués d'enviar tanta informació per segon. Tot i aquesta reducció els fluxos rebuts per l'usuari 1 seguien presentant inestabilitats i el nombre de paquets perduts en els fluxos de vídeo era molt proper al 50% del total.

Poc després l'usuari 2 va detectar que el flux de vídeo perdia masses paquets i va reduir les VDT de baixada a 96kbs i 1500kbs.

Després de aproximadament 1 minut més en que l'usuari 2 rebia correctament els fluxos de l'usuari 1, però en canvi l'usuari 1 seguia tenint problemes amb els fluxos rebuts es va reduir les VDT de baixada de l'usuari 1 a 32kbs pel flux d'àudio i 200kbs pel flux de vídeo.

Després de les tres reduccions de VDT en els diferents *peers* el numero de paquets perduts durant el que va durar la videoconferència va passar a ser de 0 en tots els fluxos rebuts per els dos usuaris i la conversa es va estabilitzar definitivament.

Videoconferència 2:

En la segona videoconferència només l'usuari 1 va reduir les VDT de baixada per sota de les inicials, ja que la velocitat de transmissió que usava l'usuari 2 per enviar els fluxos era excessiva per a ell. Al igual que l'altre cas la reducció va ser primer a 96kbs i 1500kbs i després es va passar a enviar a 32kbs pel flux d'àudio i 200kbs pel flux de vídeo.

Fins al cap de dos minuts no es van aconseguir establir les VDT que permetien l'intercanvi de fluxos fluït ja que després de fer la primera reducció de VDT va costar arribar al 50% de paquets perduts de nou.

Videoconferència 3:

Durant els 4 minuts de durada de la videoconferència no va haver-hi cap reducció de VDT.

En aquest segon *test*, per a la detecció de VDT's excessives i reducció d'aquestes, es volien realitzar més proves, però com que aquesta funcionalitat es va afegir cap al final no va haver-hi prou temps per poder construir un escenari més gran en que i participessin més de dos usuaris. Tot i així, al ser una funcionalitat a nivell de comunicació P2P, afegir més usuaris només a d'implicar repetir casos com el que s'ha testejat.

De manera menys formal es varen fer també varies proves durant la implementació de la funcionalitat i es va trobar el problema de que alguns cops, si les VDT superen per poc a les que s'haurien d'usar, no s'arriba mai a una pèrdua del 50% dels paquets enviats i la conversa entre els dos usuaris mai acaba de ser fluida. La solució a això passa per reduir el percentatge de paquets perduts permès abans no es redueixen les VDT, però al reduir el percentatge s'està exigint també una precisió que implicaria més proves de les que hem fet per assegurar que no es redueixen les VDT de canals que estan funcionant correctament.

Tots les proves de la lògica de la interfície i les velocitats de transmissió en els cinc escenaris s'han fet tant en els navegadors *Chrome* com amb *Firefox*, així que no es coneix com respondrà l'aplicació en altres navegadors. Tot i això, s'espera que els problemes que es puguin tenir altres navegadors (excepte *Internet Explorer* i *Safari*) siguin mínims i superficials.

Per altre banda, el testing de l'última funcionalitat (reducció de VDT en temps real) només s'ha provat en el navegador *Chrome* ja que per *Firefox* s'ha suposat que no funcionaria.

Després del test es pot dir que en *Chrome* es dona suport a un 99% de les funcionalitats, mentre que en Firefox, degut al fet de que el navegador no dona encara suport a la gestió dels objectes SDP, no es fa el control de les VDT, fent que a la que s'acumulen més de tres usuaris en una mateixa videoconferència el funcionament d'aquesta passi a ser un atzar. En aquest cas, per a establir videoconferències entre navegadors *Firefox* s'aconsella als usuaris utilitzar només senyal d'àudio i no de vídeo, ja que si els usuaris estan en xarxes amb velocitats de transmissió molt diferents sortirà el problema que ens passava si regulàvem malament les velocitats de transmissió en els diferents escenaris.

Excepte la regulació d'ample de banda el sistema dona suport a la resta de funcionalitats en *Firefox*.

En el cas de videoconferències entre navegadors *Chrome* i *Firefox* alhora, tots els requeriments de la videoconferència segueixen funcionant correctament tot i que el fluxos de pujada del navegador *Chrome* cap a *Firefox* (que ha de regular el navegador *Firefox*) no es regulen, i pot ser que si els dos usuaris tenen VDT diferents l'usuari que està en *Firefox* no rebi correctament les dades de l'usuari que usa el navegador *Chrome* (missatges, fitxers, fluxos multimèdia, ...).

11. Implantació i resultats

Aquest apartat va destinat a la implantació i resultats obtinguts en els diferents requisits del sistema.

IMPLANTACIÓ DEL SISTEMA

El sistema s'ha instal·lat sobre un servidor que, tot i que no té moltes prestacions, ha estat suficient per a poder utilitzar-lo amb un nombre poc elevat d'usuaris.

Desconec actualment els límits del servidor però si es tingués la intenció de fer l'aplicació pública seria necessari un canvi de servidor. En aquest aspecte, la implantació del sistema en un nou servidor és senzilla, a partir de la instal·lació d'Ubuntu 14.04.4 i les funcionalitats necessàries només cal copiar la part servidora a una carpeta del sistema i la part del client s'ha de copiar al directori al que redirecciona el servidor *Apache*.

En el projecte, a partir de la instal·lació del servidor es va procedir a implantar un sistema de videoconferències bàsic entre usuaris ubicats en xarxes diferents i llavors es va fer l'anàlisi de requeriments.

Un cop clares les funcionalitats que havia d'oferir el sistema es va implementar tota la part referent a la videoconferència (interfície més API ASWRTC), en aquest cas però es va deixar per més endavant totes les funcionalitats que no eren primordials, d'aquesta manera es volia assegurar poder complir amb els requisits que es van posar en el full de ruta abans d'implementar-ne de nous.

L'API ASWRTC en aquest primer moment permetia únicament la gestió de fluxos multimèdia i la limitació manual de les VDT dels usuaris, tot i que al igual que actualment, estava preparada per afegir noves funcionalitats.

Sabent ja que *WebRTC* funcionava correctament en tots els escenaris possibles es van implantar pràcticament tots els casos d'ús dels dos primers diagrames, es va començar pels casos d'ús més importants i un cop es tenien complerts els requisits de prioritat essencial es van unificar les parts de gestió de perfil amb les de la videoconferència.

Va ser en aquest moment quan es van implementar les funcionalitats de mitjana necessitat per a la pàgina de videoconferències (gravació, gestio de les dades gravades i xat no persistent),a part d'això es va afegir a l'API ASWRTC la funcionalitat d'obtenir i proporcionar estadístiques sobre els fluxos multimèdia enviats.

Després d'això s'ha anat millorant mica en mica la interfície de l'usuari implantant funcionalitats superficials com mostrar les videoconferències actives, soroll quan un usuari rep un missatge, avisos per a informar de peticions... En aquest aspecte es poden encara anar millorant coses respecte el control d'esdeveniments inesperats que poden llançar els usuaris i especialment en temes d'informació cap a aquests.

Com a funcionalitat d'última hora es va afegir l'opció de detectar canals (*PeerConnections*) inestables i canviar les VDT en temps real. Aquesta funcionalitat no estava prevista, però després d'haver estudiat el protocol SDP i les seves possibilitats es va valorar aquesta possibilitat de canviar les VDT sense la necessitat de tancar el *PeerConnection* actual amb l'altre usuari i crear-ne un de nou.

Inicialment es tenia previst implementar menys funcionalitats de les que s'han fet, ja que en el full de ruta es van considerar les funcionalitats de manera molt general i al definir-les aquestes tenien moltes sub-funcionalitats per a millorar la comunicació entre usuari i sistema.

La implantació de la part referent al servidor (base de dades, API REST i API Socket.IO), i en general totes les funcionalitats de Back-end, es van anar fent a mesura es necessitaven. Això va fer que el disseny inicial de la base de dades es veies afectat algun cop per necessitats del client, ja que quan es va fer el disseny inicial encara no es tenia del tot clar com es gestionarien certs requeriments del sistema (en especial la part de gestió de velocitats de transmissió).

RESULTATS FINALS

Per la part *back-end* s'han realitzat les proves descrites en l'apartat 10.2 i s'ha comprovat la correctesa de la base de dades (que es guardessin les dades de manera encriptada i correctament) a través de la interfície que proporciona ***phpMyAdmin***.

Per exemple, aquest és l'aspecte de la taula d'usuaris:

codi	nickName	correu	sexe	password	avatar_url	data
0	$mð Âˆè*ö óßÎ`	n¾óž,~éÕ ¤É¯^onâeyÜi{9"_]9-Ñ|Ü	€›@H„çpÁ ±«Õpù˜uø	$2a$10$4UPaCxSLfRtgovWbRJ6FZe34rhoApTVE619k.SZsYxJ...	}²âØGÀžwŒS9´i@c	2016-08-23 16:00:41
1	ÂX–öwß,7S°õ	Âbæ'þ,...œü $pŠÙ<Öq !ÓDÇ‡ôá°õûÂ	6âQ„RAç™¾UÖ˜YU	$2a$10$gv6UmwLTbBE9mUAQWvTDueaDmRbWlpVu.gkosKHtc7b...	*NULL*	2016-08-13 12:36:53
2	ïÏ!·[å(dÓbçL±Õ	Ŝ Đ›6Ió¢góù¶	€›@H„çpÁ ±«Õpù˜uø	$2a$10$eAJbuxbUeWcvzOXEQG1Q/.iuug.nVdPb/Qc9p3QTOyI...	¡ƒ;Ö'Ü"\zÐj¤;'	2016-08-13 12:31:22
3	àGDwEK&†½(1Żw·à	Yè¸J¯÷/ ô+jeaõq<Öq !ÓDÇ‡ôá°õûÂ	€›@H„çpÁ ±«Õpù˜uø	$2a$10$dU9xBoOmoTC697G3EOWkPeOL7.vvFQSlJxhAhy7WEwx...	^I^Þ›«ŜF`+¢zg	2016-08-23 16:02:08

Per la part visible per l'usuari, per demostrar que s'han implantat tots els casos d'usos exposats a la fase de requeriments s'han realitzat tres vídeos, un per cada diagrama de casos d'ús. Cada vídeo fa referència a una pàgina Web del sistema a la que en la fase de requeriments se li ha associat un diagrama de funcionalitats (apartat 6.3).

L'idea dels vídeos es fer una mica de guia a través de cada pàgina Web, situant els casos d'ús a la interfície del sistema i com els actors interactuen amb ells:

Pàgina principal (primer diagrama de casos d'ús en el que participa l'actor usuari):

https://www.dropbox.com/s/e4fgd7hm4jh58ui/paginaPrincipal.wmv?dl=0

Pàgina personalitzada (segon diagrama de casos d'ús en el que participen l'usuari identificat i el sistema):

https://www.dropbox.com/s/rfmvbsn3kyqth99/paginaPersonal.avi?dl=0

Pàgina de videoconferències (últim diagrama de casos d'ús per l'actor usuari trucant i el propi sistema):

https://www.dropbox.com/s/90c44yfgwr03pj0/paginaVideoconferencia.avi?dl=0

12. Conclusions

El sistema final ofereix una interfície que intercomunica en temps real, a través dels medis d'àudio, vídeo i text, tots els usuaris registrats. Per a fer això, a un usuari autentificat se li ofereix la possibilitat de crear converses amb altres usuaris, amés de la possibilitat de realitzar videoconferències entre ells.

Els resultats finals s'adhereixen a l'**abast i objectius** inicials amb els que es va proposar el treball. Tot i així penso que, després d'haver estudiat a fons la tecnologia *WebRTC*, encara es pot arribar més lluny, tal com està explicat en l'apartat de treball futur.

Respecte l'idea inicial el més complicat ha estat trobar una solució per a conèixer i restringir les velocitats de transmissió que els usuaris podien usar per a les videoconferències. En aquest cas l'idea de fer el test de velocitat basant-se en la geocalització de l'usuari ha donat millors resultats dels esperats i permet regular la qualitat de les videoconferències als recursos de xarxa dels usuaris participants.

Per evitar possibles errors en el test de velocitat s'ha proporcionat un sistema automàtic de disminució de VDT, cosa que ha permès que:

- Si el programador que assigna les VDT a l'usuari s'equivoca el sistema podrà reaccionar.
- Si la xarxa té comportaments inesperats durant un cert període de temps es pot solucionar.
- El control de la VDT es fa tenint en compte la xarxa entre els usuaris i o no el camí entre usuari i servidor.

Aquestes solucions permeten la realització de converses estables en *Chrome* de manera fiable (especialment per a ordinadors de taula) però presenta els inconvenients de que el test de velocitat és lent i que si un usuari canvia de xarxa des de la mateixa geocalització les velocitats de transmissió que s'usen passen a ser incorrectes, en aquest cas, la API ASWRTC permet solucionar els problemes d'inestabilitat però si canviem a una millor xarxa no s'augmentaran les VDT fins que es realitzi el test de velocitat de nou.

Referent a la part *front-end* en especial l'ús d'AngularJS ha permès cobrir bastant ràpidament les funcionalitats essencials, permetent així poder implementar la llista de requeriments de prioritat mitja. Però en el que millor ha ajudat AnguarJS ha estat en la modulació del codi, ja que penso que ha quedat un codi bastant entenedor per a si algun altre programador vol reutilitzar o consultar parts del codi fet.

Respecte la **planificació inicial** s'ha seguit bastant l'esquema de l'apartat quatre tot i que els temps destinats a cada funcionalitat s'han incomplert bastant. Aquest fet ha estat degut a que de totes les tecnologies i *Frameworks* utilitzats, o bé no els havia utilitzat mai o en tenia un baix coneixement. *AnguarJS* per exemple es va estudiar de nou, la implementació del servidor de *NodeJS* (tot i que en tenia coneixements teòrics) també ha estat des de zero, i per la part de WebRTC, tot i ser l'única part de la que partia amb una base, he après bastant més durant el projecte i penso que encara em queda molt per aprendre (ja que *WebRTC* lliga molts camps relacionats amb l'intercanvi massiu de dades entre navegadors).

Per altre banda, la funcionalitat de detecció i reestabliment de VDT en temps real no es tenia prevista i es va implementar a la segona quinzena d'agost un cop acabada tota la part de gestió persistent dels usuaris.

Un altre aspecte que va alterar el temps de desenvolupament va ser el fet de que tampoc tenia clar quines llibreries s'usarien pel projecte, en especial, per la part de la interfície, vaig començar comunicant les interfícies amb la lògica usant JQuery i al final, veient que amb Anguar el codi guanyava molt en aspectes de senzillesa, vaig acabar esborrant parts complexes i poc entendibles que havia fet anteriorment amb JQuery.

Pel que fa al compliment de la **metodologia de desenvolupament en cascada** hi han hagut punts en que s'ha incomplert. Durant la fase d'implementació vaig haver de refer el disseny de la base de dades i algunes crides de les API's REST i Socket.IO degut a que quan els vaig dissenyar no ho vaig fer amb les idees prou clares.

Aquest fet, en especial la part de Socket.IO ha fet que en el diagrama de classes, algunes petites parts perdin el seu sentit original, per sort però, gràcies a l'ús de patrons com l'Straegy això no va comportar també canvis en l'estructura.

En general, el projecte m'ha aportat els coneixements que esperava adquirir quan es va proposar:

- Instal·lació i gestió d'un servidor Web.
- Millores en el coneixement de llibreries i entorns de programació Web.
- Major coneixement sobre l'intercanvi massiu de dades de manera directe entre navegadors.

Pel desenvolupament i implementació del projecte ha estat molt important la base adquirida en cada una de les assignatures del grau. Com a reflexió personal, després de fer un balanç del que he après, considero que pel projecte han estat necessaris cada un dels conceptes adquirits durant la carrera, tant pel que fa el seu ús directament reflectit al treball o com a base per aprendre les noves eines que s'han usat. Sense la base que crec haver adquirit amb el grau el temps destinat a aprendre cada nova tecnologia, llibreria o entorn de treball s'hagués incrementat de manera molt notable. Per això penso que el projecte m'ha aportat nous coneixements però també crec que ha estat una bona posta a pràctica del que ja sabia.

COMPARACIÓ AMB ALTRES APLICACIONS

L'aplicació desenvolupada no està al nivell de les aplicacions més famoses, ja que s'hauria de donar servei a les plataformes mòbils com fan la majoria d'aplicacions d'alt prestigi, cosa a que no es fa actualment.

Respecte això, el problema que presenta l'API ASWRTC és que està feta en *JavaScript* i es per la part del client. Això implica que per implementar-la en diferents plataformes s'ha de traduir al llenguatge de programació que usa la plataforma. En aquest aspecte però, no cal dissenyar res de nou i l'únic que cal és traduir el codi de la l'API (les llibreries de WebRTC estan disponibles en múltiples llenguatges).

Malgrat això per la part web, en especial per *Chrome*, gràcies a que les qualitats de la videoconferència són adaptables a les de la xarxa en la que es troba l'usuari, el sistema de videoconferència proporcionat permet evitar molt efectivament les comunicacions inestables.

D'altre banda, es dona la possibilitat de que si l'usuari detecta converses inestables pot parar el vídeo o àudio per reduir els fluxos de dades que envia, o si no té cap dispositiu per a transmetre senyals sempre pot entrar en una videoconferència com a oient i participar a través del xat.

Una de les principals utilitats per la quals es va construir aquest sistema va ser amb la idea de millorar el sistema de videoconferències VITAM. Els problemes més importants de VITAM són la seva dependència amb LICODE i la falta de control de les velocitats de transmissió per a videoconferències en P2P.

La API ASWRTC és una alternativa per VITAM a l'ús de LICODE. En aquest aspecte, si VITAM passes a usar la API aquesta li proporcionaria una millora respecte les videoconferències P2P proporcionant-li totes les funcionalitats que la API presenta, i de les que actualment no disposa:

- *Obtenció d'estadístiques per a canals P2P.*
- *Regulació de les velocitats de transmissió de manera automàtica.*
- *Gravació amb preprocés instantani.*
- *Possibilitat d'enviament de dades (fitxers, imatges i missatges) de manera directe.*

13. Treball futur

Per portar SRTM al nivell dels grans sistemes de comunicació cal **estendre la seva implantació als dispositius mòbils**. Aquest fet implica implementar primer les APIS ASSIO i ASWRTC per als sistemes operatius d'Android i IOS. Si això es fes caldria traduir les APIs ASWRTC i ASSIO a *Java* (per Android) i *ObjectiveC* (per IOS).

Un aspecte més pesat seria la implementació de tota la part referent a la gestió d'usuaris en Android i IOS, ja que la construcció d'interfícies sempre s'acaba fent llarg.

Centrant-nos en les funcionalitats de la API ASWRTC, que es l'encarregada de proporcionar totes les funcionalitats referents a la gestió de fluxos multimèdia i de dades en una videoconferència, **es podria treballar més la gestió de velocitat de transmissió en temps real per tal de que no fos necessari un test de velocitat inicial**. Això implica establir un protocol entre els dos usuaris d'un PeerConnection per trobar el més ràpid possible les VDT que més els i convenen per a cada flux multimèdia. Aquest protocol haurà de permetre tant la disminució com l'augment de les VDT usades de manera independent per a cada flux que s'envia.

Una altre funcionalitat que es podria afegir a l'API ASWRTC és **la possibilitat de la regulació de les VDT usades per enviar fluxos de vídeo en el navegador *Firefox***. Encara que *Firefox* no doni suport a la regulació de les VDT a nivell de xarxa (no permet modificar l'objecte SDP com en *Chrome*) es podrien regular les qualitats del vídeo en el moment en que indiquem que volem obtenir dades de la càmera de l'usuari.

Quan es crida el mètode *getUserMedia()* per obtenir les dades del dispositiu se li pot indicar la resolució de les imatges que s'obtindran de la càmera. Això permet que després de fer un estudi de les comprensions que realitza el còdec que s'utilitza per a codificar i descodificar els *streams* de vídeo alhora d'enviar-les a través de la xarxa, es podria fer el procés invers al que es realitza. És a dir, en comptes de dir-li al còdec que té *x bitrate* per codificar i enviar una imatge de la millor qualitat possible, es podria fer la reducció de resolució de la imatge en el moment en que es recuperada i no imposar-li cap limitació al còdec.

Tenint en compte això, es podria fer una llista de possibles resolucions d'imatge i depenent de les VDT que doni el test de velocitat per a l'usuari escollir millors o pitjors resolucions.

Aquesta és la única manera que actualment podríem utilitzar per regular les VDT en els fluxos sortints d'un navegador *Firefox*. Però tot i que teòricament pot tenir sentit podria no funcionar, ja que mentre es feien proves durant el procés d'implementació del sistema es va observar que encara que limitis la resolució de la càmera, s'envien més o menys dades depenent de les velocitats de transmissió que es permetin, és a dir, si en dos usuaris es limiten les resolucions de la seva càmera a 640x480 píxels per imatge processada, però s'assignen diferents velocitats de transmissió pel flux de vídeo, l'usuari amb més VDT envia més bits per segon pel flux de vídeo que l'altre, tot i que en principi no sembli necessari. Per això si es vol implementar aquesta funcionalitat s'hauria de mirar realment quin és el comportament de *WebRTC* en aquest aspecte.

Un altre problema de la regulació de VDT a través de la qualitat del vídeo recuperat en *Firefox* és que a diferència de *Chrome*, si es volguessin canviar les VDT un cop s'ha establert una *PeerConnection* amb un altre usuari, es molt provable que fos necessari tancar la connexió i obrir-ne una de nova amb les noves resolucions. Això és degut a que cal parar l'obtenció de dades del

dispositiu de vídeo i indicar-li que ara es recuperaran les dades amb una altre resolució; Aquest procés implica un canvi en l'objecte *MediaStream* que li fa canviar la URL única amb la que els altres el veuen.

Com a última cosa a millorar **a nivell d'interfície, s'haurien de proporcionar als desenvolupadors gràfics sobre les VDT usades durant tota una videoconferència**. Actualment la part d'obtenció d'estadístiques de ASWRTC funciona correctament, però la informació es mostra a través de la consola. El que s'hauria de fer és construir gràfics en la part del servidor informat de com van funcionat les videoconferències, per així, en el cas de que els usuaris tinguin problemes, el desenvolupador sàpida a que han estat deguts.

Per que no s'han implementat aquestes funcionalitats i s'han deixat pel futur:

- La idea de fer una aplicació multiplataforma ha estat pel temps, per a expandir SRTM a Android i IOS seria necessàries unes 300 hores més, ja que en *Objective C* encara no hi he programat mai.

- Proporcionar a la API ASWRTC la possibilitat de gestionar la VDT en *Firefox* no ha estat tant pel temps sinó per la falta de coneixement. Fins que no vaig conèixer bé tot el procés d'obtenció i enviament de vídeo i un cop fetes les proves per a la limitació de qualitats a través de les velocitats de transmissió no se'm va acudir aquesta possible solució per a *Firefox*. A més, implementar aquesta funcionalitat implicaria també un llarg procés de proves en diferents escenaris per veure'n el seu efecte.

- La funcionalitat d'afegir un control de VDT independent al test de velocitat (també per a l'API ASWRTC) no s'ha fet perquè cal un procés de proves molt més extens del que s'ha fet per tal de proporcionar una gestió de VDT en temps real adequada i eficient. Aquesta funcionalitat podria fins i tot implicar que la API ASWRTC necessités una BD per guardar i obtenir estadístiques.

- Per a proporcionar als desenvolupadors estadístiques sobre la VDT usada el temps de feina que comportaria es més depenent a la qualitat de la interfície que es vulgues oferir. En aquest cas s'haurien d'enviar la variació de les VDT usades per a cada *MediaStreamTrack* i *PeerConnection* al servidor al final d'una videoconferència, o cada un cert període de temps, i guardar-ho a la base de dades per poder-ho consultar en qualsevol moment.

14.Bibliografia

Apis WebRTC

[1] http://www.tutorialspoint.com/webrtc/webrtc_media_stream_apis.htm

[2] https://w3c.github.io/webrtc-pc/#rtcpeerconnection-interface

[44] https://w3c.github.io/webrtc-stats/#bib-RFC3550

Protocol STUN

[3] http://www.viagenie.ca/publications/2008-09-24-astricon-stun-turn-ice.pdf

[4] http://www.3cx.com/blog/voip-howto/stun-details/

[5] https://tools.ietf.org/html/rfc3489

[6] https://msdn.microsoft.com/en-us/library/dd946184%28v=office.12%29.aspx

[7] https://tools.ietf.org/html/rfc5766#page-37

Trickle ICE Candidates

[8] https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection/canTrickleIceCandidates

Socket.IO en Java

[9] https://github.com/socketio/socket.io-client-java

[10] https://github.com/mrniko/netty-socketio/tree/master/src

Session description protocol (SDP)

[11] https://en.wikipedia.org/wiki/Session_Description_Protocol

[12] https://webrtchacks.com/anatomy-webrtc-sdp/

Protocols WebRTC

[13] http://webrtc-security.github.io/

[45] https://tools.ietf.org/html/rfc3550

Moduls NodeJS

[14] https://nodejs.org/api/https.html#https_https

[15] https://nodejs.org/api/fs.html#fs_file_system

[16] https://github.com/mysqljs/mysql

[17] http://www.nodehispano.com/2012/01/express-el-framework-web-para-nodejs/

[18] https://www.npmjs.com/package/body-parser-json

[19] https://github.com/caolan/async/blob/v1.5.2/README.md

[20] http://codetheory.in/using-the-node-js-bcrypt-module-to-hash-and-safely-store-passwords/

Instal·lació NodeJS

[21] https://www.digitalocean.com/community/tutorials/how-to-install-node-js-on-an-ubuntu-14...

Llibreries usades per a la API sobre WebRTC

[22] https://github.com/muaz-khan/RecordRTC

[23] https://www.webrtc-experiment.com/RecordRTC/

[24] https://github.com/muaz-khan/getStats

[25] https://webrtcexperiment-webrtc.netdna-ssl.com/BandwidthHandler.js

[26] https://github.com/muaz-khan/DetectRTC

[27] https://www.webrtc-experiment.com/DetectRTC/

Frameworks i mòduls usats per al control de la interfície

[28] https://jquery.com/

[29] https://angularjs.org/

[31] https://github.com/grevory/angular-local-storage

[32] http://www.w3schools.com/html/html5_geolocation.asp

Bits per segon i còdecs

[30] https://en.wikipedia.org/wiki/Bit_rate

Protocol HTTP

[33] http://www.tutorialspoint.com/http/http_methods.htm

Xifrat de dades

[34] http://dev.mysql.com/doc/refman/5.7/en/encryption-functions.html#function_aes-encrypt

Llibre Real-Time Communication with WebRTC

[35] http://droppdf.com/v/ZEPXg

Exemple WebRTC de manera local

[36] https://github.com/webrtc/samples/tree/gh-pages/src/content/peerconnection/pc1

Certificats SSL per HTTPS

[37] https://www.digitalocean.com/community/tutorials/how-to-secure-apache-with-let-s-encrypt-on-ubuntu-14-04

[38] https://www.digicert.com/ssl-certificate-installation-apache.htm

Peticions HTTPS des de dominis diferents al del servidor

[39] https://www.youtube.com/watch?v=cUWcZ4FzgmI

Exemples peticions asíncrones

[40] http://justinklemm.com/node-js-async-tutorial/

[41] https://baudehlo.com/2014/04/28/node-js-multiple-query-transactions/

[42] http://stackoverflow.com/questions/31799175/async-waterfall-in-a-for-loop-in-node-js

[43] http://stackoverflow.com/questions/30642055/mysql-and-node-async-waterfall

Instal·lació STUN/TURN

[46] https://groups.google.com/forum/#!msg/easyrtc/ypjJ5Yu3wZM/u5Lq6VNfabcJ

15. Annexos

15.1. Extensió dels requisits funcionals

En aquesta secció es mostra la llista de tots els requisits funcionals que el sistema ha d'oferir als usuaris:

Registrar usuari

Un usuari ha de poder registrar-se a l'aplicació entrant el seu nom identificatiu, correu, sexe (el sexe només s'usa per a dibuixar els avatars (home o dona) fins que l'usuari no escull una foto de perfil) i contrasenya.

Nom del requisit	Registrar usuari		
Font del requisit	Usuari		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Identificar usuari

Un cop registrat, autentificar un usuari a partir del seu nom identificatiu i contrasenya.

Nom del requisit	Identificar usuari		
Font del requisit	Usuari		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Llistar usuaris

Mostrar i buscar un usuari, a partir del nom, per a poder-li enviar un petició d'amistat.

Nom del requisit	Llistar usuaris		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Enviar petició d'amistat

Poder seleccionar un usuari buscat i enviar-li una petició d'amistat.

Nom del requisit	Enviar petició amistat		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Mostrar peticions

Mostrar les peticions rebudes i enviades a altres usuaris permetent poder-les seleccionar segons el nom amb un *input d'autocomplete*.

Nom del requisit	Llistar peticions conversa		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Respondre peticions

Un cop rebuda una petició d'una conversa poder-la acceptar i passar a poder-la usar o rebutjar-la.

Nom del requisit	Respondre petició conversa		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Canviar configuració trucada

Permetre la selecció del tipus de funcionalitats que vol tenir en la videoconferència de la conversa, permetent seleccionar si vol enviar l'àudio, vídeo o permetre gravar i ser gravat.

Nom del requisit	Canviar configuració trucada		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Llistar amics

Llistar tots els amics que té l'usuari, podent seleccionar els amics que comencen amb determinats noms.

Nom del requisit	Llistar amics		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Seleccionar amics

Seleccionar una llista d'amics determinats per incloure'ls en una petició de conversa múltiple. També ha de permetre, un com un amic ha estat seleccionat, poder-lo esborrar de la llista de seleccions.

Nom del requisit	Seleccionar amics		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Enviar petició conversa

És una funcionalitat molt semblant a la d'enviar petició d'amistat, però en aquest cas la petició que s'envia és per una conversa en la que participen múltiples usuaris. Aquests usuaris als que se'ls i envia la petició tenen la restricció de que han de ser amics de l'usuari que envia la petició.

Nom del requisit	Enviar petició conversa		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Seleccionar conversa

D'entre totes les converses de l'usuari, permetre-li seleccionar una conversa determinada i posar-la en primer pla, mostrant els missatges rebuts i donant l'opció d'enviar missatges de text, fitxers, imatges i poder establir una videoconferència.

Nom del requisit	Seleccionar conversa		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Tancar conversa

Amagar la conversa que s'està mostrant en primer pla, ja sigui perquè ha clicat la creu de sortir o perquè ha seleccionat una altre conversa per posar en primer pla.

Nom del requisit	Tancar conversa		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Llistar converses en les que participa l'usuari

Mostrar totes les converses de l'usuari (perquè les pugui seleccionar) juntament amb tots els missatges guardats d'aquella conversa, perquè quan seleccioni la conversa pugui veure els missatges enviats i rebuts antiguament.

Nom del requisit	Llistar converses		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Enviar trucada

Enviar un avis a tots els usuaris d'una conversa perquè acceptin o no entrar en la videoconferència que l'usuari vol fer. Els usuaris que reben l'event poden acceptar o rebutjar l'opció d'entrar a la trucada. Si almenys un usuari accepta la petició, s'obre la videoconferència amb els usuaris que l'han acceptat.

Nom del requisit	Enviar Trucada		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Enviar missatge

Enviar, un cop oberta una conversa, un missatge de text a tots els usuaris participants a la conversa seleccionada. Un cop enviat el missatge ha de mostrar-se en el xat de rebuda de tots els usuaris.

Nom del requisit	Enviar Missatge		
Font del requisit	Usuari identificat		
Prioritat del requisit	☒Alta/Essencial	☐Mitja/Desitjat	☐ Baixa/Opcional

Enviar arxiu

Enviar, un cop oberta una conversa, un arxiu, que pot ser una imatge o un fitxer, a tots els usuaris participants a la conversa seleccionada. Un cop enviat l'arxiu ha de mostrar-se en el xat de rebuda de tots els usuaris. En el cas de ser un fitxer s'ha de permetre descarregar-lo a través d'un enllaç, una imatge, d'altre banda es mostra directament.

Nom del requisit	Enviar Arxiu		
Font del requisit	Usuari identificat		
Prioritat del requisit	☐Alta/Essencial	☒Mitja/Desitjat	☐ Baixa/Opcional

Eliminar una conversa

L'usuari pot eliminar una conversa de la que forma part. En el cas de que la conversa esborrada sigui de només 2 usuaris (ell i un altre) també s'esborra de la llista d'amistats, l'amistat entre els 2 usuaris.

Nom del requisit	Esborrar conversa		
Font del requisit	Usuari identificat		
Prioritat del requisit	☐Alta/Essencial	☒Mitja/Desitjat	☐ Baixa/Opcional

Entrar en una videoconferència ja activa

Permetre a l'usuari entrar en una videoconferència activa d'una conversa de la que forma part. Aquesta funcionalitat ha de permetre a l'usuari entrar a una videoconferència a la qual ha arribat tard o n'ha sortit un moment.

Nom del requisit	Entrar conversa trucada		
Font del requisit	Usuari identificat		
Prioritat del requisit	☐Alta/Essencial	☒Mitja/Desitjat	☐ Baixa/Opcional

Modificar perfil

L'usuari ha de poder canviar les dades de registre inicials, essencialment el seu nom, correu i foto de perfil.

Nom del requisit	Modificar perfil		
Font del requisit	Usuari identificat		
Prioritat del requisit	☐ Alta/Essencial	☒Mitja/Desitjat	☐ Baixa/Opcional

Calcular amples de banda

Calcular la velocitat de transmissió (ample de banda) de pujada i baixada que te la xarxa en la que es troba l'usuari. Gràcies això es permet conèixer quins són els límits de velocitats de transmissió a les que es pot arribar en una videoconferència, per tal de no sobrepassar-los i sobrecarregar la xarxa.

Nom del requisit	Modificar perfil		
Font del requisit	Sistema		
Prioritat del requisit	☒ Alta/Essencial	☐ Mitja/Desitjat	☐ Baixa/Opcional

Control de posició de l'usuari

Aquesta funcionalitat permet al sistema no calcular cada cop l'ample de banda d'un usuari, la funcionalitat calcula les coordenades (Latitud,Longitud) a les quals es troba l'usuari i busca si mai havia estat en aquelles coordenades. Si hi havia estat suposa que esta connectat a la mateixa xarxa i no re-calcula els amples de banda de pujada i baixada de l'usuari.

Nom del requisit	Ubicar usuari		
Font del requisit	Sistema		
Prioritat del requisit	☐ Alta/Essencial	☒ Mitja/Desitjat	☐ Baixa/Opcional

Comprovar connectivitats

Saber quan tots els usuaris d'una conversa estan connectats. La funcionalitat que llança el sistema permet mostrar al sistema els usuaris que estan en línia i els que no.

Nom del requisit	Comprovar connectivitats		
Font del requisit	Sistema		
Prioritat del requisit	☐ Alta/Essencial	☒ Mitja/Desitjat	☐ Baixa/Opcional

Les funcionalitats que es mostren a continuació són únicament per la part de videoconferència.

Gestionar dades locals

Aquesta funcionalitat és una composició de funcionalitats sobre la gestió que se li ofereix a l'usuari per al control de els senyals que vol enviar als altres usuaris. La funcionalitat permet parar/engegar el vídeo que s'envia, parar/engegar la senyal d'àudio que s'envia, gravar/parar de gravar-se a si mateix.

Nom del requisit	Gestionar usuari local		
Font del requisit	Usuari trucant		
Prioritat del requisit	☒ Alta/Essencial	☐ Mitja/Desitjat	☐ Baixa/Opcional

Gestionar dades remotes

Aquesta funcionalitat és també una composició de funcionalitats, però sobre la gestió que se li ofereix a l'usuari per al control de els senyals que rep dels altres usuaris. El requeriment permet seleccionar un usuari remot per a parar/engegar el vídeo que veiem, parar/engegar l'àudio que escoltem, gravar/parar de gravar els seus fluxos multimèdia.

Nom del requisit	Gestionar usuari local		
Font del requisit	Usuari trucant		
Prioritat del requisit	☒ Alta/Essencial	☐ Mitja/Desitjat	☐ Baixa/Opcional

Enviar missatge no persistent en videoconferència

Enviar dins de la videoconferència un missatge de text, no persistent en futures sessions, a tots els usuaris participants a la videoconferència. Un cop enviat el missatge ha de mostrar-se en el xat de rebuda de tots els usuaris.

Nom del requisit	Enviar Missatge videoconferència
Font del requisit	Usuari trucant
Prioritat del requisit	☒Alta/Essencial ☐Mitja/Desitjat ☐ Baixa/Opcional

Enviar arxiu no persistent en videoconferència

Enviar, un cop dins de la videoconferència, un arxiu no persistent en futures sessions (que pot ser una imatge o un fitxer) a tots els usuaris participants a la conversa seleccionada. Un cop enviat l'arxiu ha de mostrar-se en el xat de rebuda de tots els usuaris. En el cas de ser un fitxer s'ha de permetre descarregar-lo a través d'un enllaç, una imatge, d'altre banda es mostra directament.

Nom del requisit	Enviar Arxiu videoconferència
Font del requisit	Usuari trucant
Prioritat del requisit	☐Alta/Essencial ☒Mitja/Desitjat ☐ Baixa/Opcional

Gestionar velocitats de transmissió

Detectar canals *WebRTC* amb *MediaStreamTracks* que envien més dades per segons de les que les VDT de les xarxes on estan ubicats els dos usuaris permeten. Un cop detectat que hi ha un o més *MediaStreamTracks* amb connexions inestables reduir els seus *bitrates* d'enviament a la VDT que permet la xarxa per tal d'estabilitzar-los i recuperar la fluïdesa en l'intercanvi de fluxos.

Nom del requisit	Gestionar VDT
Font del requisit	Sistema
Prioritat del requisit	☐Alta/Essencial ☐Mitja/Desitjat ☒ Baixa/Opcional

15.2. Diagrames de seqüència

Els diagrames de seqüència per a cada un dels casos d'ús són els següents. Els diagrames és mostren agrupats per les pàgines Web a les que pertanyen i no és mostra més enllà de les classes frontera de les API's ASWRTC i ASSIO; això es perquè si un nou programador vol implementar un sistema de videoconferències usant la nostre API no ha de saber ni quines classes estan per sota les classes frontera.

En els mètodes per a la comunicació entre classes no es mostren els paràmetres per no fer molt complexes els diagrames, en el codi, hi han les especificacions d'entrada i sortida a sobre de la definició de cada mètode.

DIAGRAMA SEQÜÈNCIA PÀGINA INICIAL:

Diagrama seqüència cas d'us Registrar Usuari:

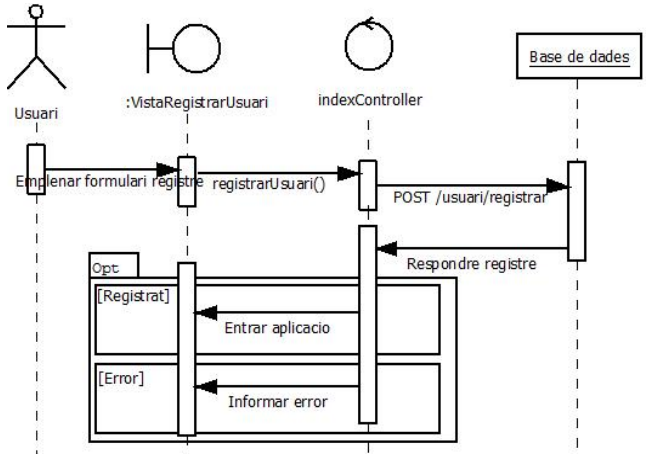


Diagrama seqüència cas d'us Identificar Usuari:

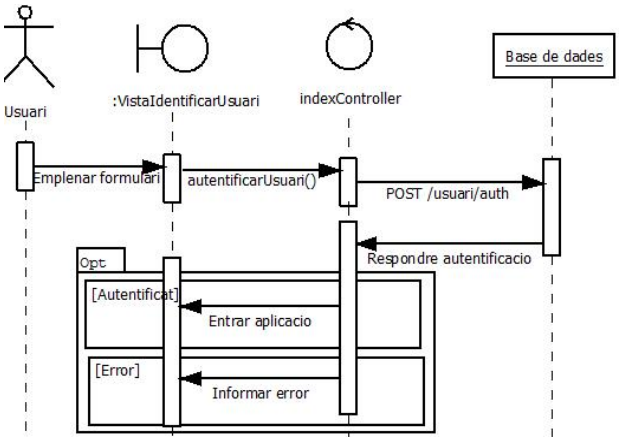


DIAGRAMA SEQÜÈNCIA DE LA PÀGINA PERSONALITZADA PELS USUARIS :

Diagrama seqüència cas d'us enviar arxiu:

L'usuari envia un arxiu (imatge o fitxer) a través del xat persistent.

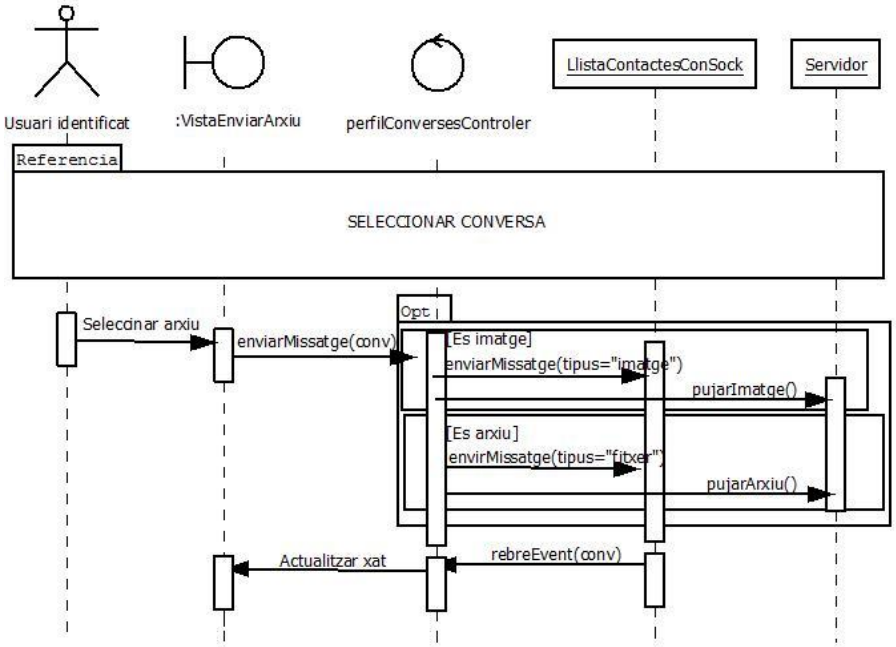


Diagrama seqüència cas d'us Modificar Perfil:

L'usuari modifica el seu perfil, pot modificar correu, nom identificatiu i foto de perfil.

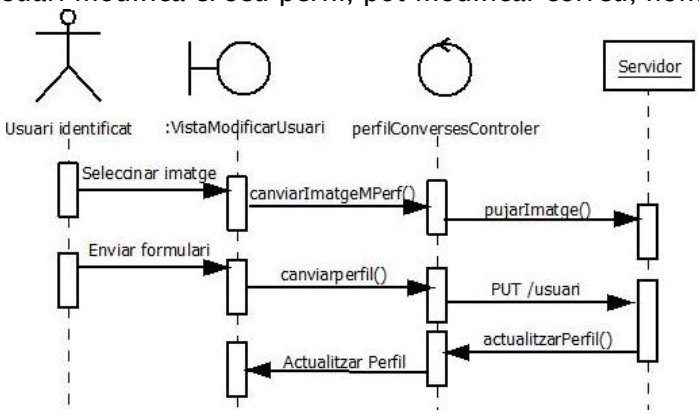


Diagrama seqüència cas d'us Llistar Usuaris:

Es llisten tots els usuaris que compleixen amb la busqueda de les lletres entrades per l'usuari.

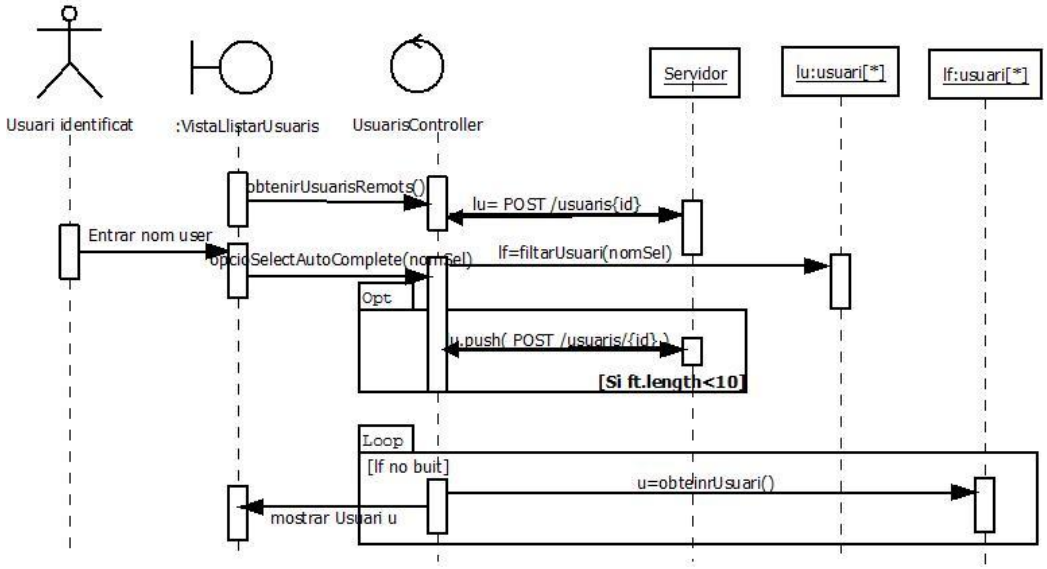


Diagrama seqüència cas d'us Enviar petició conversa:

L'usuari selecciona varis amics i els i envia una petició de conversa múltiple.

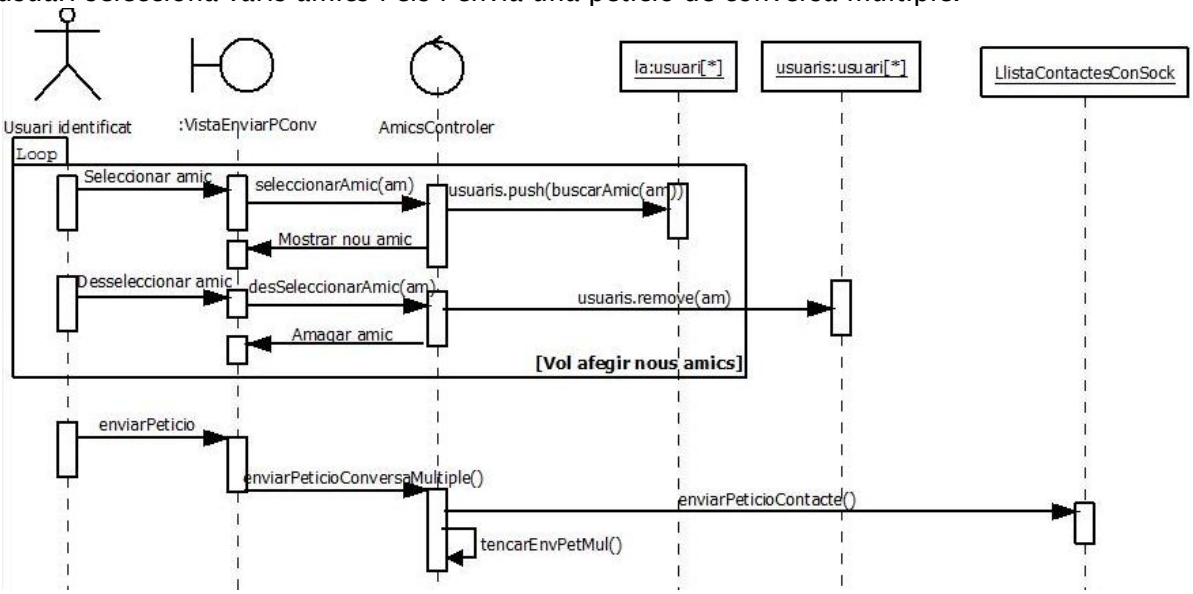


Diagrama seqüència cas d'us Enviar petició amistat:

L'usuari selecciona un usuari de la vista d'usuaris i li envia una petició d'amistat.

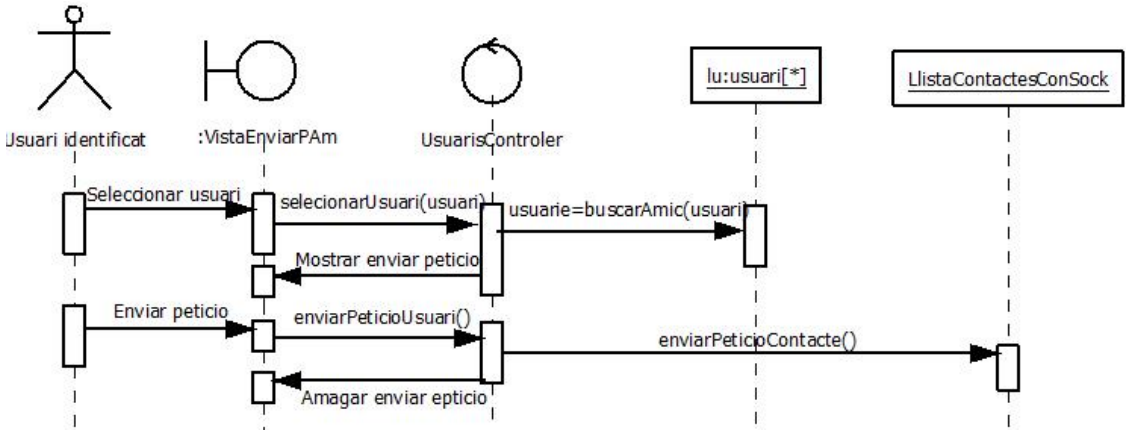


Diagrama seqüència cas d'us Llistar peticions conversa:

Es llisten totes les peticions de conversa que s'han fet o s'han rebut i que contenen el *substring* entrat en el nom.

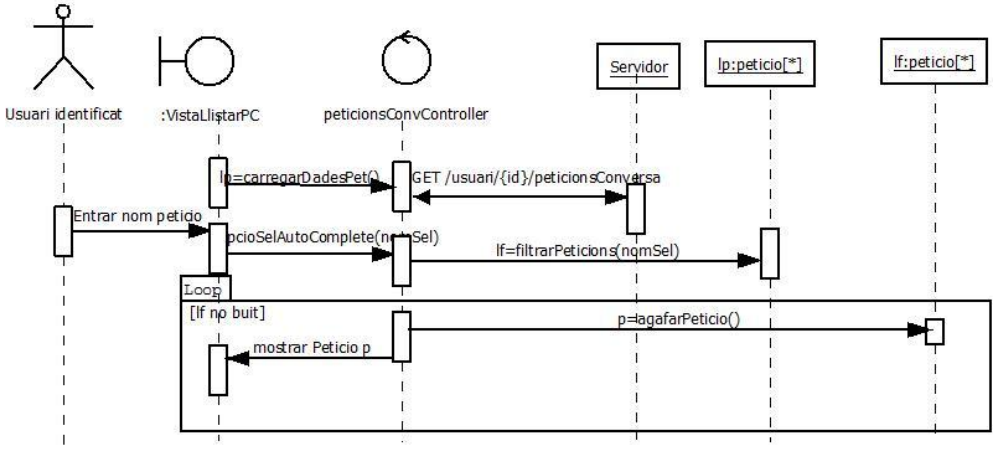


Diagrama seqüència cas d'us Resposta petició conversa:

L'usuari respon una petició de conversa que li han fet, primer la selecciona i després la respon.

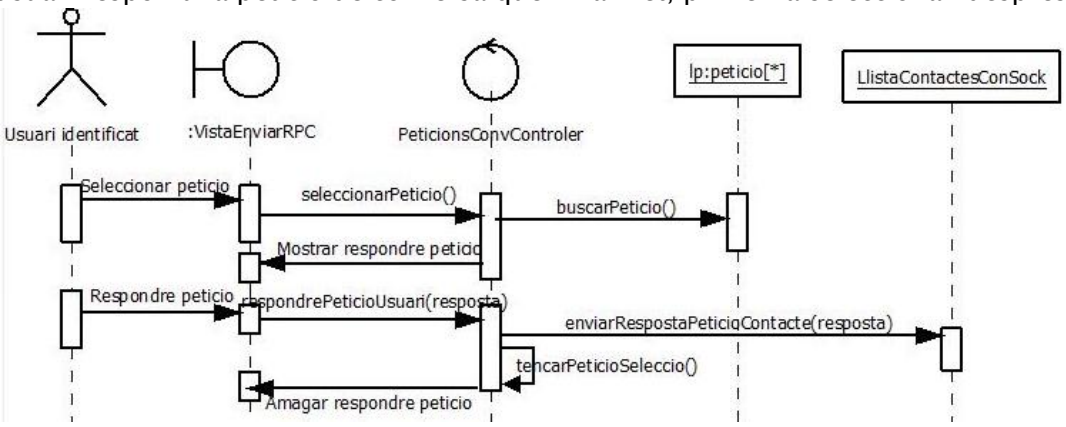


Diagrama seqüència cas d'us Llistar Amics:

L'usuari entra un mot i es mostren tots els amics que contenen aquell mot en el seu nom.

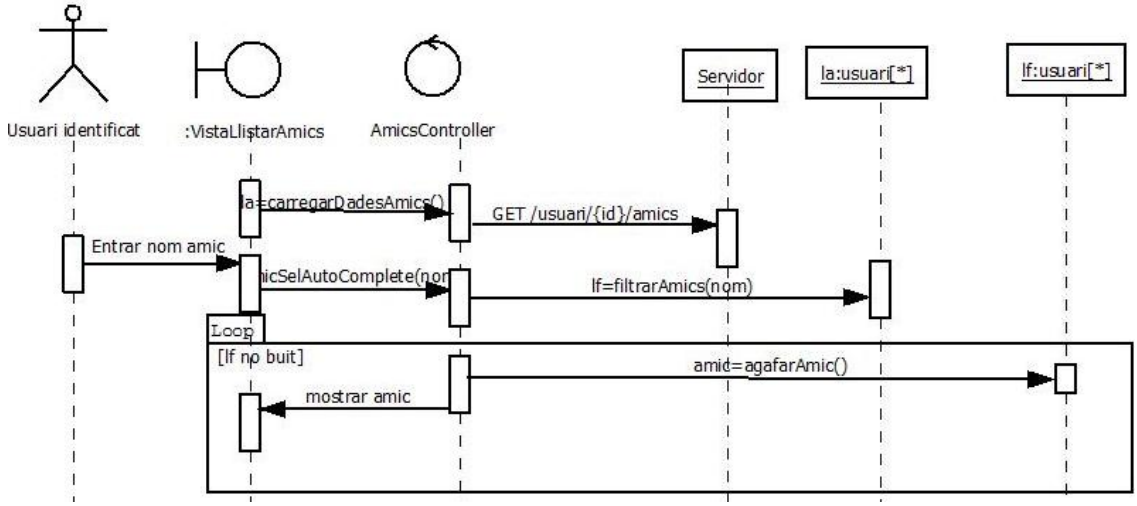


Diagrama seqüència cas d'us Llistar converses:

Es mostren totes les converses de l'usuari perquè les pugui obrir.

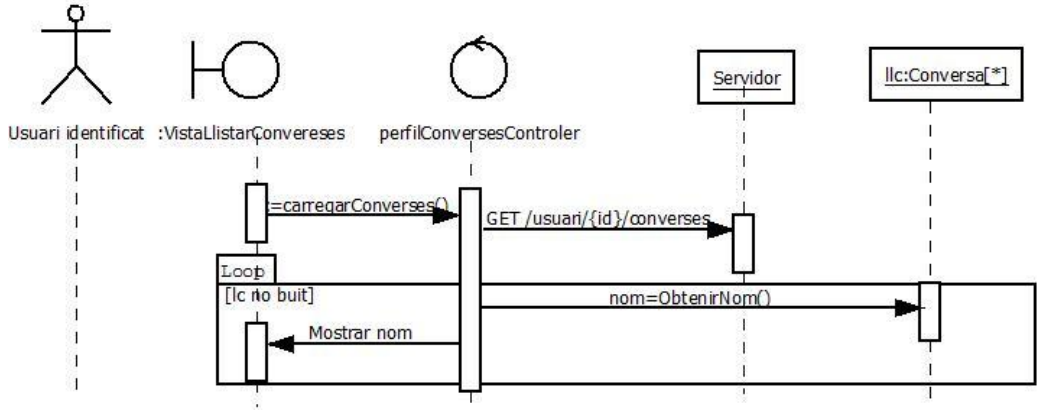


Diagrama seqüència cas d'us Enviar missatge:

L'usuari envia un missatge de text a través del xat persistent.

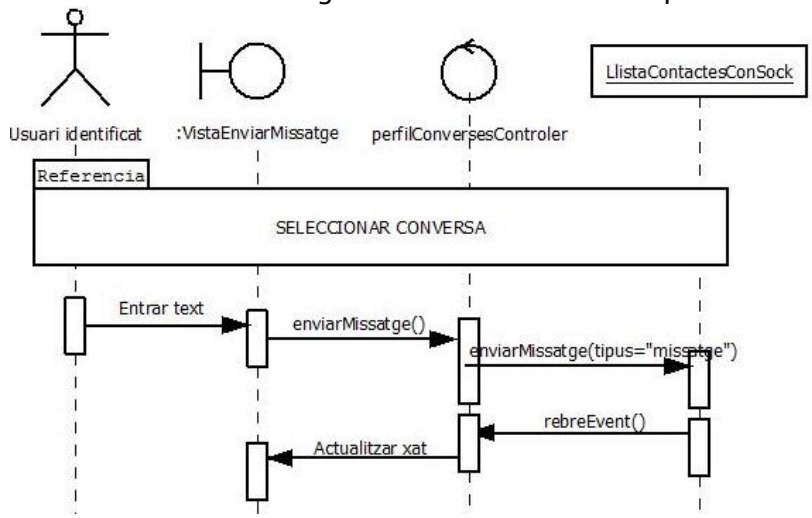


Diagrama seqüència cas d'us Enviar trucada:

L'usuari envia una trucada per establir una videoconferència entre els usuaris d'una conversa.

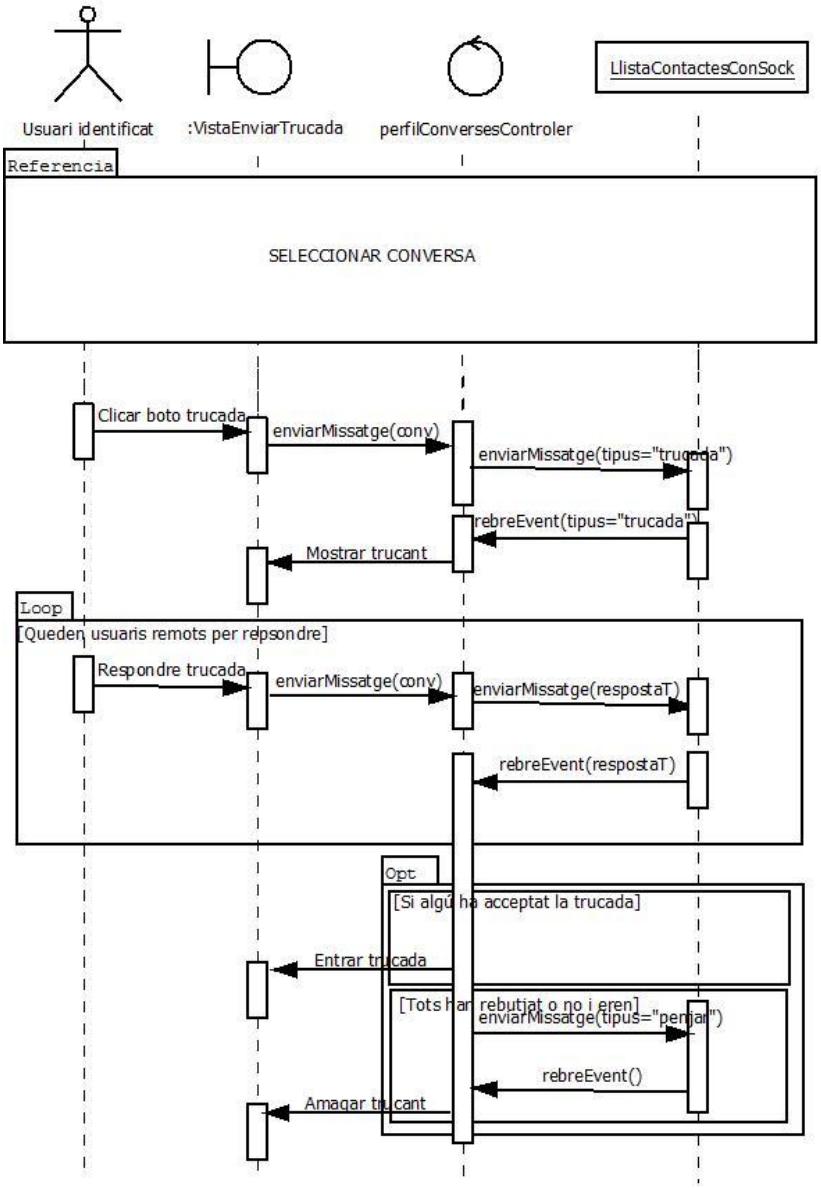


Diagrama seqüència cas d'us Esborrar Conversa:

S'elimina la conversa seleccionada per l'usuari, a la resta d'usuaris també se'ls hi esborra.

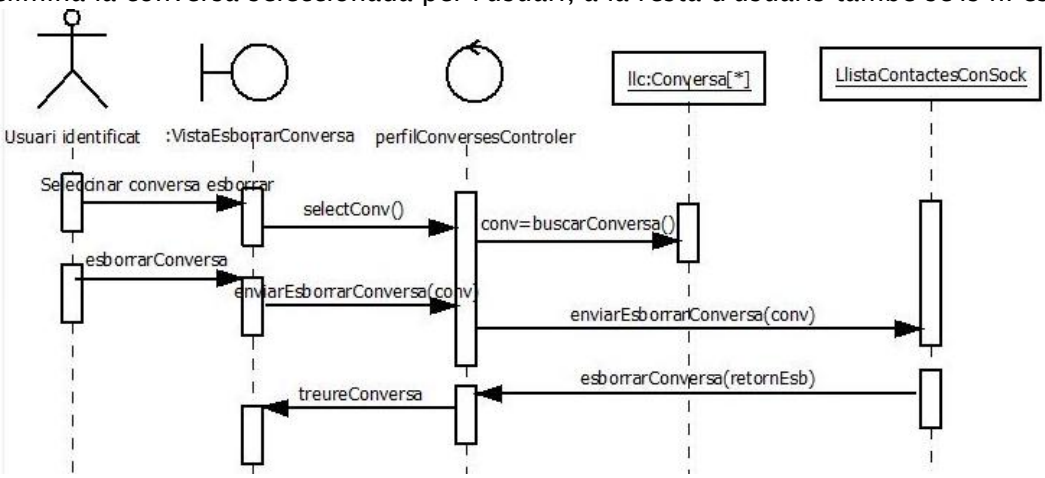


Diagrama seqüència cas d'us tancar Conversa:

Es desselecciona la conversa que estava seleccionada.

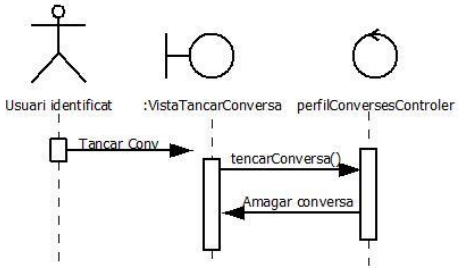


Diagrama seqüència cas d'us Seleccionar Conversa:

Es selecciona una de les converses llistades i es fa gran.

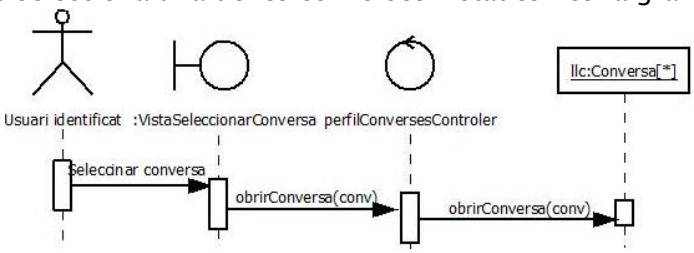


Diagrama seqüència cas d'us Mostrar Converses Actives:

Es mostren totes les converses en les que s'està realitzant una videoconferència.

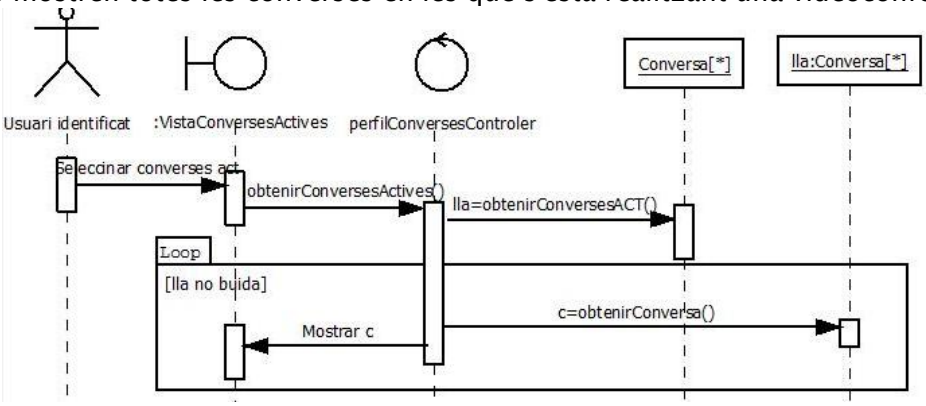


Diagrama seqüència cas d'us Entrar Conversa Trucada:

S'entra a la videoconferència d'una conversa.

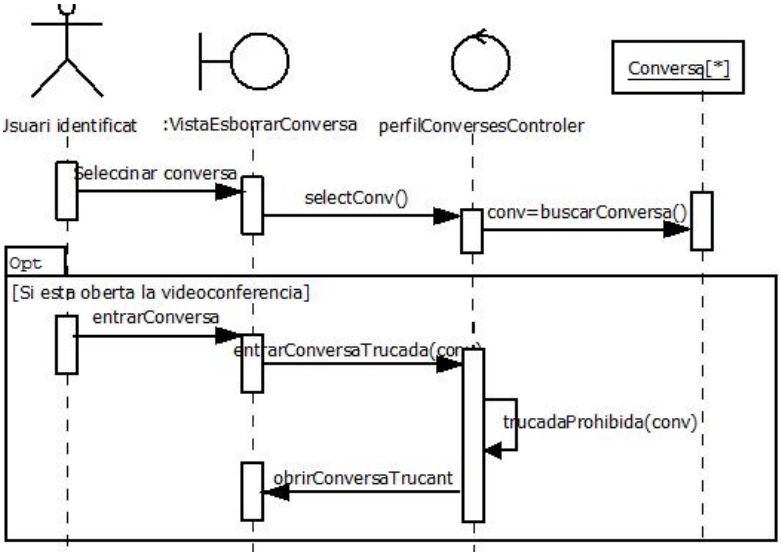


Diagrama seqüència cas d'us Canviar configuració Trucada:

L'usuari canvia les opcions de la videoconferència (àudio, vídeo i gravació).

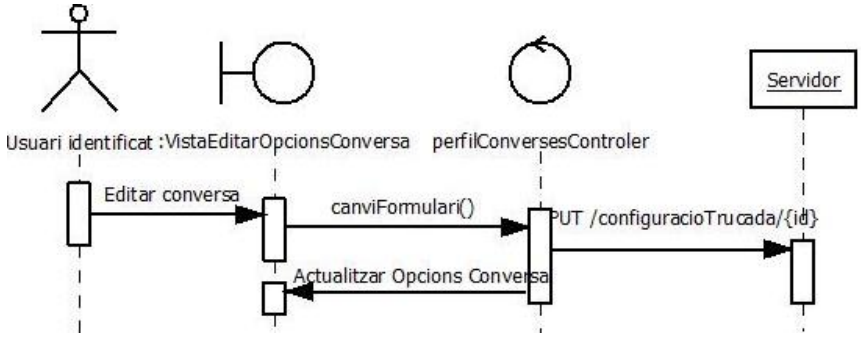


Diagrama seqüència cas d'us Ubicar usuari i Calcular amples de banda:

El sistema (en *background*) analitza si es necessari calcular els amples de banda de l'usuari o ja ho havia fet. Si no els havia calculat els calcula, sinó repaprofita els amples de banda antics.

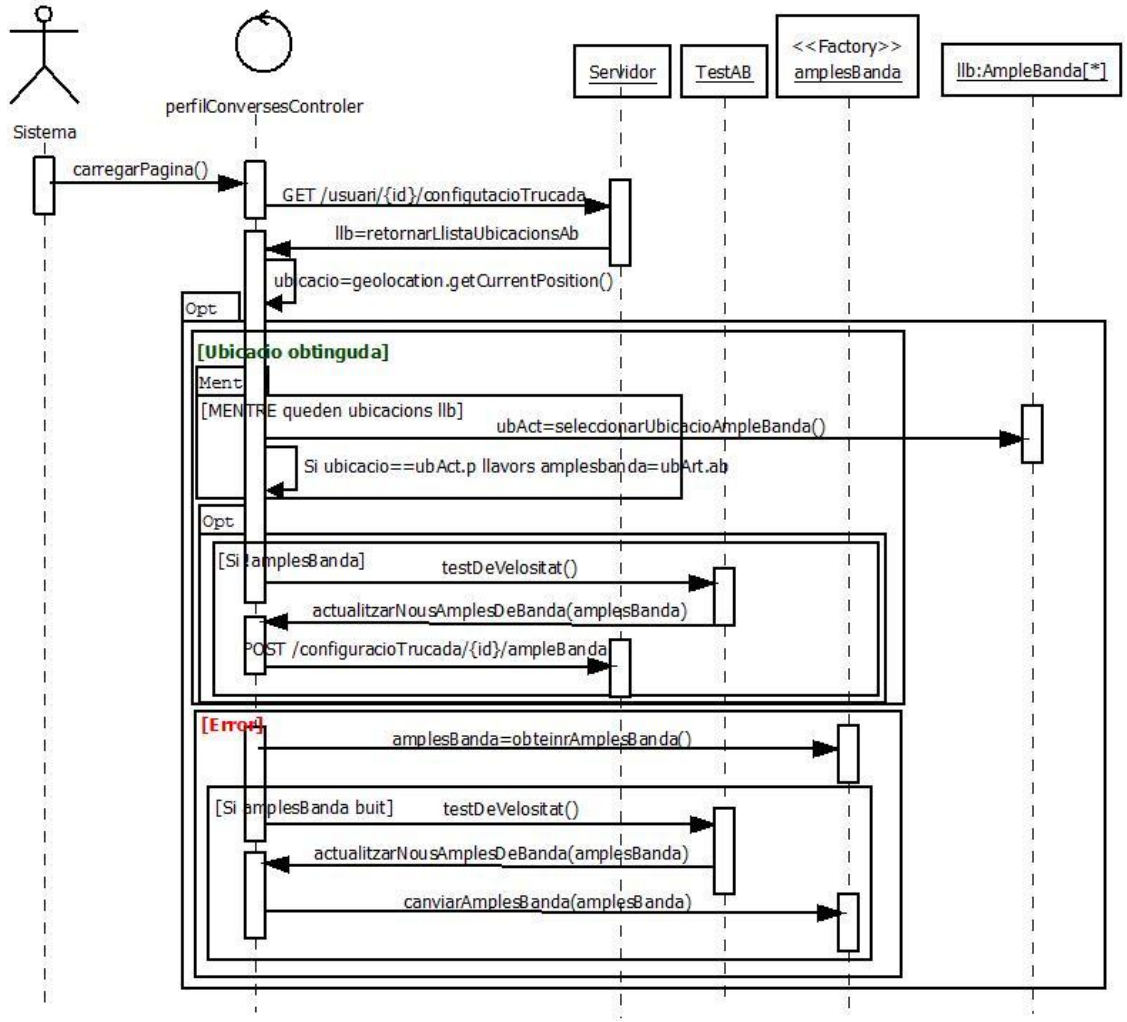


Diagrama seqüència cas d'us Comprovar connectivitats:

El sistema comprova per a cada conversa, si tots els usuaris estan en línia.

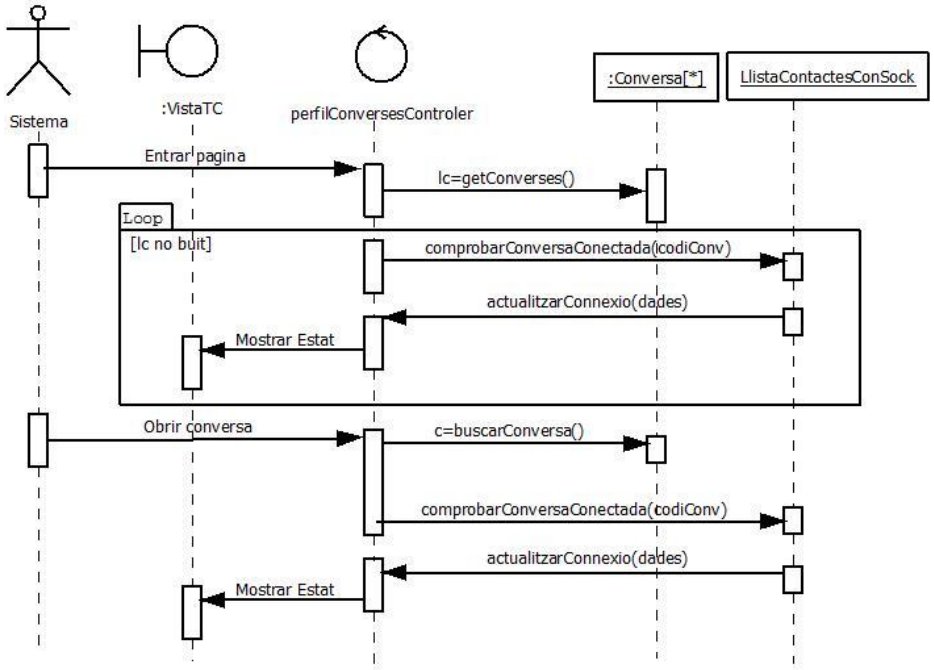


DIAGRAMA SEQÜÈNCIA PÀGINA VIDEOCONFERÈNCIA :

Diagrama seqüència cas d'us enviar arxiu videoconferència:

L'usuari envia un arxiu (imatge o fitxer) a través del xat no persistent de la videoconferència.

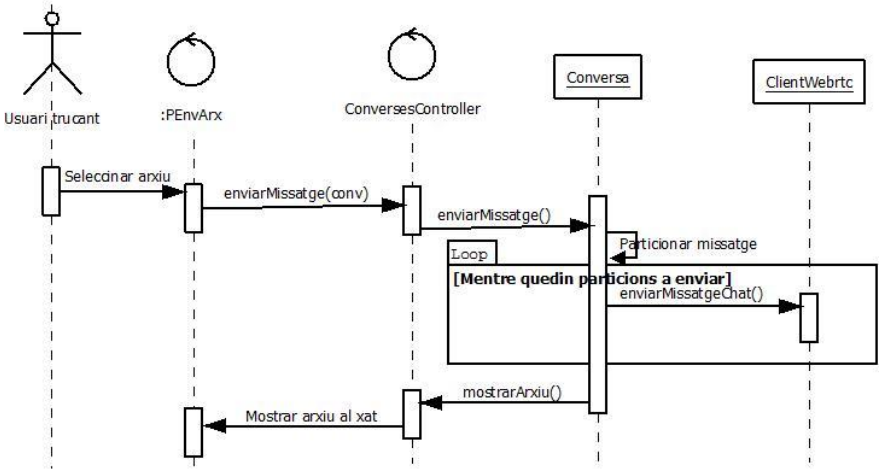


Diagrama seqüència cas d'us enviar missatge videoconferència:

L'usuari envia un missatge de text a través del xat no persistent de la videoconferència.

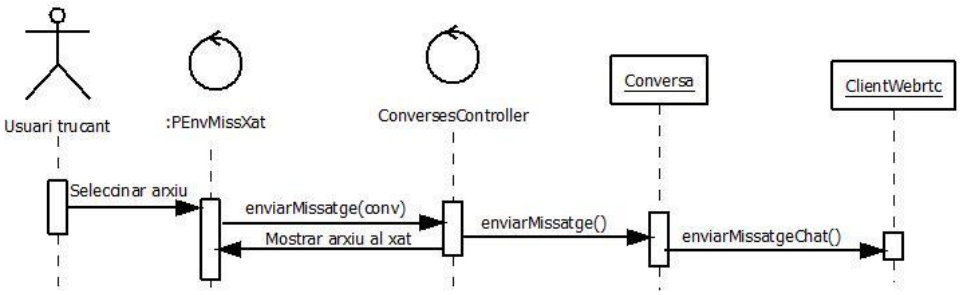


Diagrama seqüència casos d'us Gestionar usuari Local i Gestionar usuari remot:

Aquest cas d'ús es la composició de varis ja que tots segueixen la mateixa seqüència de mètodes i l'únic que canvia són els paràmetres que no es mostren, per exemple, la diferencia entre parar el vídeo local i parar el vídeo remot en la seqüència de mètodes que es segueixen és que l'argument identificador que necessiten les funcions és diferent; tot lo demés és el mateix.

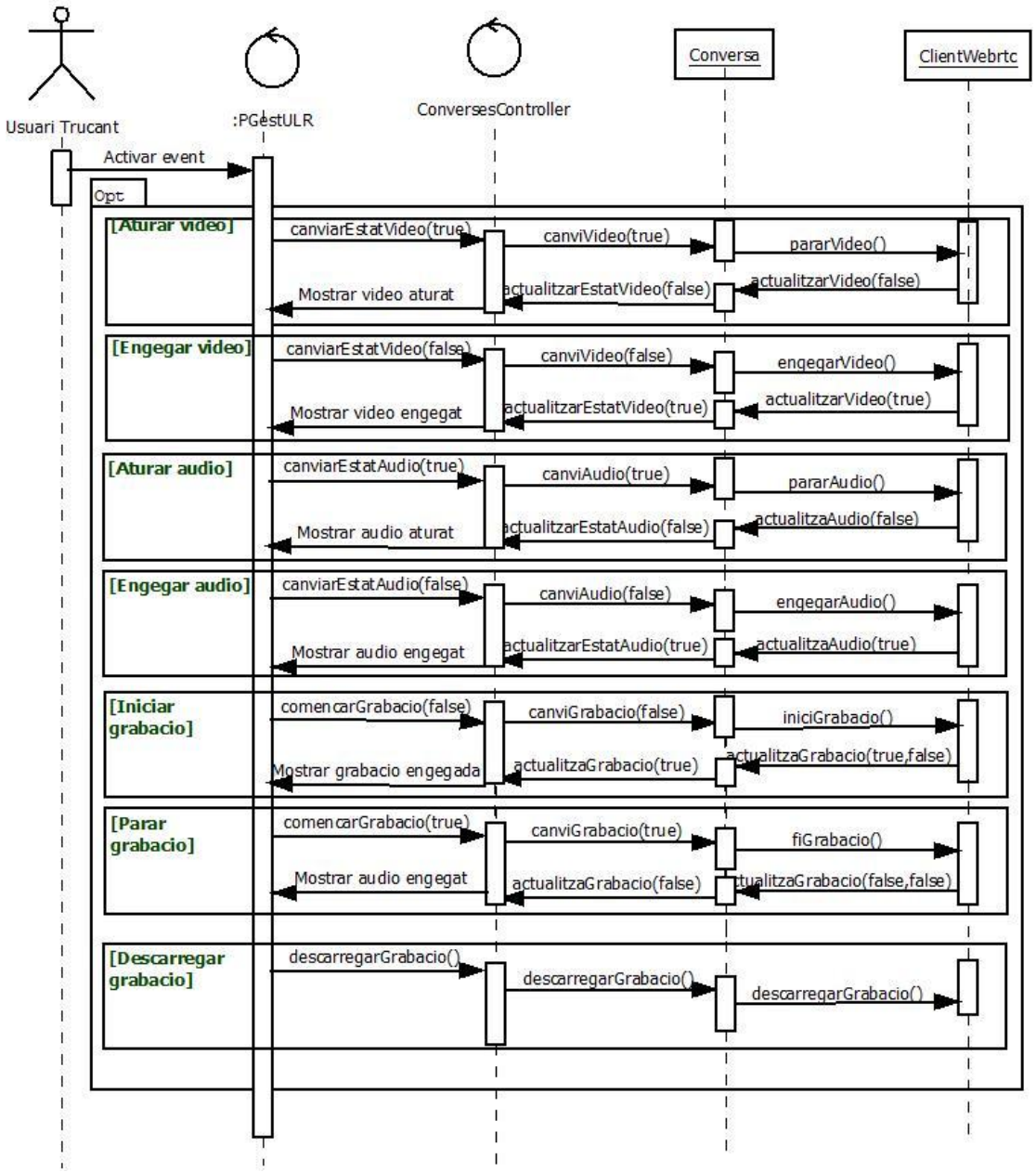
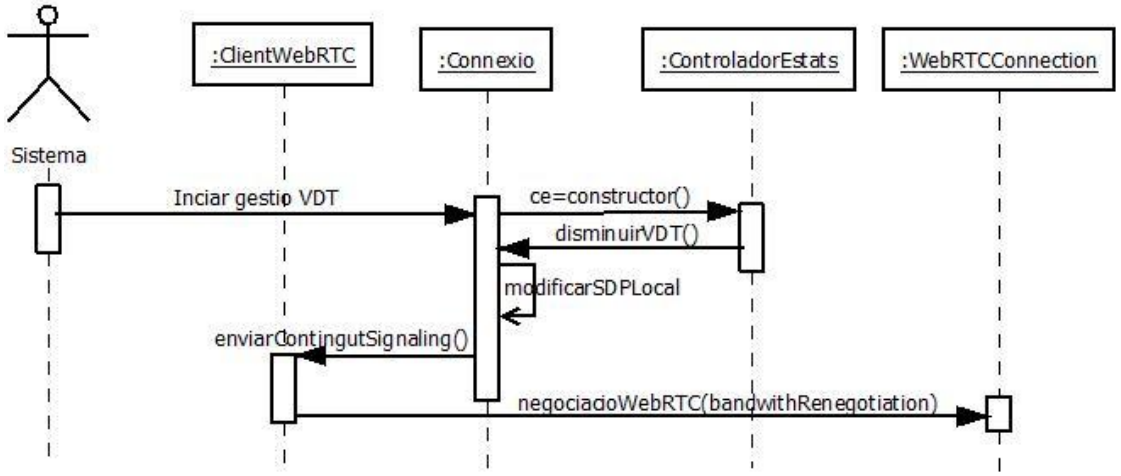


Diagrama seqüència cas d'us :

Aquest cas d'ús és l'encarregat de detectar fluxos multimèdia rebuts inestables i canviar les VDT usades pels *streams* d'entrada del *PeerConnection*.

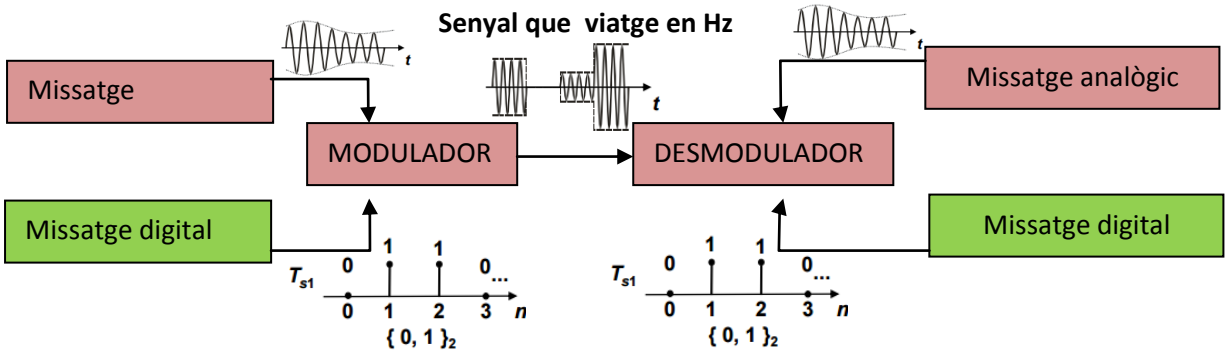


15.3. Limitació de la velocitat de transmissió

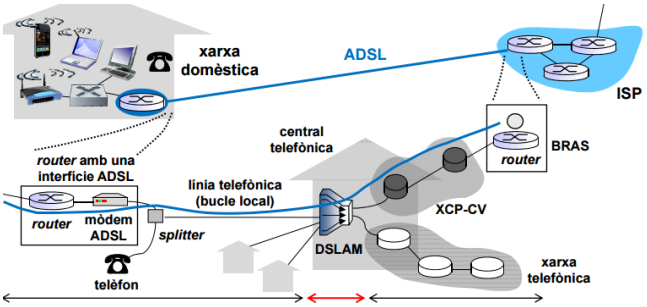
Els fluxos multimèdia que s'evien a través d'Internet no són més que una seqüència de missatges digitals (recordem que tenim missatges analògics i digitals, els analògics són de veu i en general dades que varien de manera continua i els digitals són de bits (els d'ordinador)). La quantitat d'informació que s'envia a través de la xarxa en cada segon és el que s'anomena VDT, i es mesura en ***bits per segon***. Per tal de no col·lapsar els mitjans de comunicació amb l'enviament de més dades de les que suporten, les companyies proveïdores d'Internet limiten aquesta velocitat de transmissió als seus clients.

Independentment de si el senyal que s'envia és digital o analògic (es calcula la capacitat en Hz) un cop han passat pel modulador d'entrada al canal de comunicació, els senyals analògic i digital ja són exactament iguals, per això també es diu que ens limiten és l'ample de banda (Hz) en el cas de tenir missatges digitals.

Així tenim:



Quan es modula el missatge a una senyal aquest va a parar al router i al ***mòdem*** (un mòdem ADSL (per exemple) és el que uneix el router amb la línia de telefonia), llavors via ADSL (per exemple) va cap a la central de telefonia, en el mateix cable que la veu del telèfon, però en una freqüència diferent.



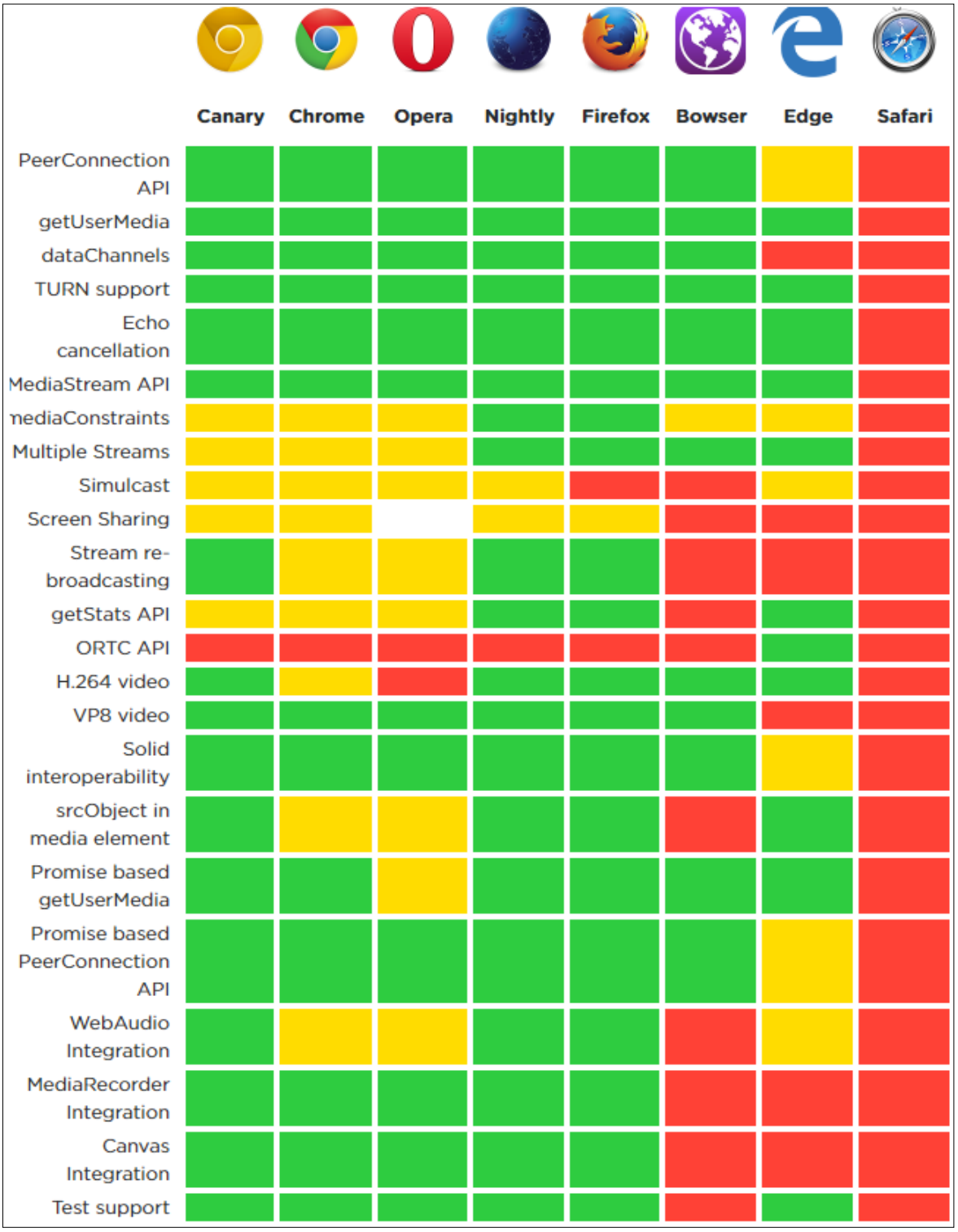
En aquest viatge fins la central telefònica la línia ADSL ja limita a 24Mbps de baixada i 2.5Mbps de pujada, cosa que significa que un usuari que està en una xarxa domestica usant ADSL ja no podrà enviar fluxos de vídeo amb qualitats superiors a HD.

Un cop la senyal arriba al DSLAM (multiplexor que connecta totes les línies ADSL del proveïdor i separa per cada línia la veu i les dades per enviar-les per camins diferents), es fa una altre limitació de l'ample de banda amb el que podem enviar i rebre senyals a través d'Internet (que pot ser menor o no a la que es tenia en la línia telefònica de bucle local).

D'aquesta manera l'ample de banda calculat en el test que es realitza a la xarxa de l'usuari és el menor entre la limitació del canal de bucle local i el que hem pactat amb la companyia.

15.4. Suport de la llibreria WebRTC

El suport actual que ofereixen els navegadors respecte la API de *WebRTC* és mostra en la següent imatge:



http://iswebrtcreadyyet.com/

Aquesta imatge va molt bé per a donar una idea de totes les funcionalitats que *WebRTC* acabarà tenint per a tots els navegadors.

En verd es presenten les funcionalitats que estan totalment implantades, en groc les funcionalitats que estan implementades en part, per exemple, per la compartició de pantalla, tant en *Chrome* com en *Firefox* la funcionalitat es dibuixa en groc pel fet de que es necessita una extensió per a que funcioni correctament. I en vermell estan pintades les funcionalitats les quals no se'ls i dona suport.

16. Manual d'usuari i instal·lació

En aquest apartat es mostra un manual tant per l'usuari com per la instal·lació del sistema en un servidor.

16.1. Manual d'usuari

Com a manual d'usuari es poden consultar els següents vídeos, que tot i que s'han fet per a la demostració de la implantació de totes les funcionalitats del sistema també serveixen com a guia per a un nou usuari:

Pàgina principal d'autentificació i registre

https://www.dropbox.com/s/e4fgd7hm4jh58ui/paginaPrincipal.wmv?dl=0

Pàgina personalitzada pel perfil de cada usuari.

https://www.dropbox.com/s/rfmvbsn3kyqth99/paginaPersonal.avi?dl=0

Pàgina de videoconferència per a una conversa

https://www.dropbox.com/s/i3njiw99etisfd3/paginaVideoconferenciaUs.avi?dl=0

Per accedir a la plana principal, si pot accedir a través de les següents URL:

https://kidd.udg.edu/

http://kidd.udg.edu/

Les dos direccions porten allà mateix, ja que en el cas d'entrar a través de la segona URL el servidor redirigeix l'usuari a la primera.

Per poder accedir al sistema és necessari crear una nova compta a través del SIGN IN i acceptar les condicions d'ús respecte el tractament de les dades personals.

Al registrar-se ja s'accedeix directament a la pàgina personalitzada. Un cop a dintre ja es poden enviar i rebre peticions de converses amb un o múltiples contactes tal com s'indica en el segon vídeo. Per enviar una petició de conversa a un usuari és necessari anar a la vista d'usuaris  , buscar l'usuari que es desitja i clicar a sobre seu per a enviar-li la petició.

Per poder enviar una conversa múltiple en la que intervenen *n* usuaris cal tenir com amics a tots els usuaris als que s'envia la petició. Per enviar la petició cal accedir a la vista d'amics  i seleccionar tots els amics que volem que participin a la conversa.

L'altre icona per a la gestió de contactes és  la d'acceptació i informació de peticions rebudes i enviades. En el cas de tenir noves peticions pendents aquesta icona es mostra amb un fons vermell.

A part d'això les demés funcionalitats del sistema són bastant lògiques i no cal explicar-les.

16.2. Manual d'instal·lació del servidor

Per la instal·lació del servidor s'han seguit els següents passos:

1. INSTAL·LAR UBUNTU

1- Instal·lar la versió més nova d'Ubuntu. Ens baixem un ISO amb la versió (que es una copia virtual d'un CD). Un cop descarregat el ISO cal que el posem en un CD per usar-lo, però hi han llocs on podem passar el ISO a tota la informació que es generaria al fer una copia a un CD i així doncs, farem això i copiarem les carpetes generades en un pen drive.

2- Iniciar sessió amb l'ordinador a on hem de fer la instal·lació i obrir la BIOS per iniciar sessió des del PEN.

3- La instal·lació és anar fent el que et demana excepte en un apartat en que et demana que es seleccioni el software que es vol instal·lar per extendre les funcionalitats del sistema bàsic. En aquest cas s'ha seleccionat:

- OpenSSH server
- DNS server
- LAMP server
- Descktop gmond (o qualsevol altre interfície per comunicar-nos amb el servidor).

2. INSTAL·LACIÓ DE PHPMYADMIN

En aquest pas instal·lem phpMyAdmin per poder visualitzar, modificar afegir i esborrar taules i entitats a la base de dades a través de la seva interfície:

sudo apt-get install phpmyadmin

És important que un cop instal·lat *phpMyAdmin*, aquest s'instal·la en una ruta diferent a la que per defecte tenim les carpetes del servidor *Apache* i cal crear un enllaç simbòlic des de les carpetes públiques d'*Apache* cap a la ruta a on està instal·lat *phpMyAdmin*. En el nostre cas s'ha usat la comanda:

sudo ln -s /usr/share/phpmyadmin /var/www/html

A partir d'aquí, per consultar la base de dades només hem d'anar al navegador i posar la ***kidd.udg.edu/phpMyAdmin*** i ja s'accedeix a la pàgina d'autentificació de phpMyAdmin.

3. INSTAL·LACIÓ DELS CERTIFICATS SSL

Per a permetre la realització i resposta de peticions del servidor Apache amb el protocol HTTP segur és necessari instal·lar un certificat que permeti l'encriptació i desencriptació de les dades enviades. Per instal·lar-los cal executar les següents comandes:

```
sudo apt-get update
sudo apt-get install git
sudo git clone https://github.com/letsencrypt/letsencrypt /opt/letsencrypt
cd /opt/letsencrypt
./letsencrypt-auto --apache -d udg.edu
```

Un cop executades les comandes es pot trobar el certificat generat a la carpeta /etc/letsencrypt/live.

4. CONFIGURAR APACHE

Cal redirigir les peticions HTTP a HTTPS i importar els certificats SSL al fitxer de configuració d'*Apache* segur (default-ssl.conf).

5. INSTAL·LACIÓ DE NODE.JS [21]

Per instal·lar un servidor NodeJS cal només executar les següents comandes:

```
sudo apt-get update
sudo apt-get install nodejs
sudo apt-get install npm
```

Llavors es poden anar afegint nous moduls amb la comanda *npm install "nomDelModul"*.

Un cop afegides totes aquestes funcionalitats al servidor *Ubuntu* ja es tenen tots els recursos necessaris per implementar el codi del sistema. Però **per a la part de WebRTC és necessari instal·lar un servidor STUN i un servidor TURN.**

6. INSTAL·LACIÓ DEL SERVIDOR STUN/TURN [46]

La instal·lació dels dos servidors s'ha de fer a un *host* que tingui disponibles dos adreces IP i dos ports visibles des d'Internet (si els volem instal·lar junts).

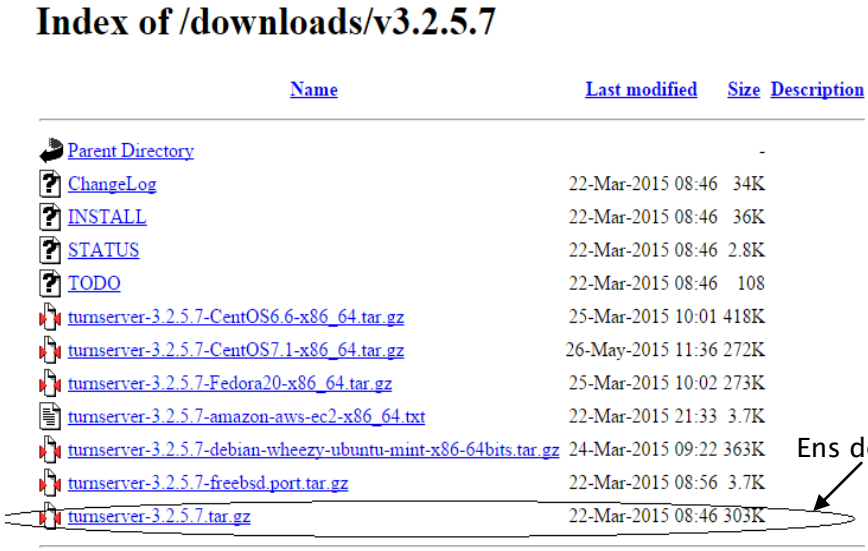
Per fer la instal·lació cal descarregar-se de la següent plana el ZIP que conté les dades del servidor:

La pàgina és la que trobem a google si posem download Turn Server (apartat downloads):

https://code.google.com/p/rfc5766-turn-server/wiki/newDownloadsSite?tm=2

I el ZIP que ens podem descarregar pot ser aquesta versió de 32 bits o una de més actual:

http://turnserver.open-sys.org/downloads/v3.2.5.7/



Un cop descarregat el ZIP caldrà que anem a la carpeta a on s'ha guardat i executem les següents comandes:

1- *tar -zxvf Downolads/turnserver-3.5.2.7.tar.gz* //Descomprimim l'arxiu comprimit i se'ns crearà una carpeta que es dirà ***turnserver-3.5.2.7/***

2- *cd turnserver-3.5.2.7/* //Entrem a la carpeta que s'ha creat al descomprimir l'arxiu ***.tar.gz***

3- *sudo ap-get update* //Actualitzem les llibreries de Ubuntu del nostre ordinador.

4- INTSTAL·LEM ELS SEGÜENTS PAQUETS:

4.1- *sudo apt-get install libssl-dev*

4.2- *sudo apt-get install libevent-2.0-5*

4.3- *sudo apt-get install libevent-dev*

4.2- *sudo apt-get install libmysqlclient-dev* (opcional)

5- EXECUTEM L'INSTAL·LADOR DE L'ARXIU DESCOMPRIMIT:

5.1- *sudo ./configure*

5.2- *sudo make*

5.3- *sudo make install*

6- PERSONALITZEM L'ACCÉS AL SERVIDOR TURN:

6.1- *cd /usr/local/etc* //Accedim a aquesta carpeta on hi han guardats els fitxers de configuració del servidor STUN+TURN i a on podrem canviar l'usuari i password del servidor TURN.

6.2- *sudo cp turnserdb.conf.default turnuserdb.conf* //Creem el fitxer ***turnusedb.conf*** que és una copia del fitxer ***turnserdb.conf.default***.

6.3- MODIFIQUEM EL FITXER ***turnserdb.conf*** PER TAL DE POSAR UN USUARI I CONTRASENYA AL SERVIDOR TURN PER RISTRINGIR L'ACCÉS A TOTHOM:

6.3.1- ***sudo vim turnserdb.conf***

6.3.2- Afegim la línia **usuari:pass** al final de tot del fitxer

Si posem això al final del fitxer, estem indicant que l'usuari és **usuari** i la contrasenya és **pass**, això significa que tot usuari que vulgui iniciar una sessió de conversa al servidor TURN (per tal de poder-se comunicar amb un altre usuari que està darrera d'un router NAT) haurà de posar aquest usuari i contrasenya per poder utilitzar el servidor TURN.