0.1 Introduzione

WebRTC è una tecnologia OpenSource, disponibile su tutti i moderni browser, che consente di effettuare videochat P2P in tempo reale.

A supporto di tale scelta tecnologica si è deciso di utilizzare il WebRTC Server Janus, in particolare di utilizzare il plugin "**VideoRoom**" di Janus che fornisce l'implementazione di una **SFU** (**Selective Forwarding Unit**).

Per quanto riguarda il client, Janus fornisce delle API in javascript (janus.js) che astraendo i dettagli implementativi della tecnologia consentono al client di interfacciarsi con semplicità ai servizi offerti da Janus.

0.2 Interazione con Janus

Il Server Janus è stato installato e mandato in esecuzione su una macchina virtuale con sistema operativo Ubuntu 16.04 su Azure ed il suo servizio è disponibile sulla porta 8088.

La comunicazione tra il Client e il Server Janus avviene tramite protocollo **WebSocket** oppure, nel caso in cui quest'ultimo non sia disponibile, tramite **HTTP** (Long Poll). Questi protocolli permettono di avere un canale bidirezionale in modo tale che il client possa ricevere eventi asincroni.

Di seguito sono mostrati alcune problematiche alle quali si è andato incontro durante la progettazione dell'interazione con il server Janus.

0.2.1 Gestione degli accessi al Server Janus

Inizialmente, nell'architettura prescrittiva, si era deciso di fare in modo che l'interazione con Janus, per quanto riguarda la creazione e l'eliminazione di una Room, fosse limitata solo al Web Application Server Smart Learning.

Tuttavia, dopo uno studio di fattibilità, si è constatato che tale scelta implementativa avrebbe richiesto di potere eseguire codice Javascript nell'application server per fare uso della libreria "janus.js" messa a disposizione da Janus.

Non essendo possibile eseguire codice Javascript su Tomcat, si potrebbe delegare tale onere a un altro server, come nodeJS.

In ogni caso tale soluzione richiederebbe un grande spreco di risorse, significherebbe mantere attivo in modo continuativo un ulteriore server per ospitare un handler per comunicare Janus.

Tuttavia, una soluzione più semplice, che in questo momento non è stata adottata ed è rimandata alla prossima release, è quella di permettere l'accesso alla funzionalità di creazione e di eliminazione di una room, non all'Application Server, ma ai docenti per un certo periodo di tempo tramite dei Ticket di servizio. In tal caso, l'Appication Server fungerebbe da fornitore dei token per poter accedere a tali funzionalità.

0.2.2 Gestione degli accessi alla Room

U'ulteriore problematica riguarda la gestione degli accessi alla room una volta creata, che consiste nel limitare l'accesso alla room ai soli studenti prenotati e al docente.

A tal fine la sequenza di passi eseguite da un docente per avviare una videocall è la seguente:

- Egli contatta l'Application Server per ottenere un nome della Room (univoco) e un insieme di token (il suo e quello degli studenti prenotati).
- Il Docente (Client Side) a quel punto, in modo automatico, si occuperà lui stesso di andare a istanziare una nuova Room, fornendo a Janus il nome della Room e i token, e configurando la stanza in modo tale da essere l'unico utente privilegiato, quindi in grado di poterla eliminare e fare il "kickout" degli studenti.
- Una volta creata la Room, egli vi accede con una semplice operazione di "Join", fornendo il proprio token.

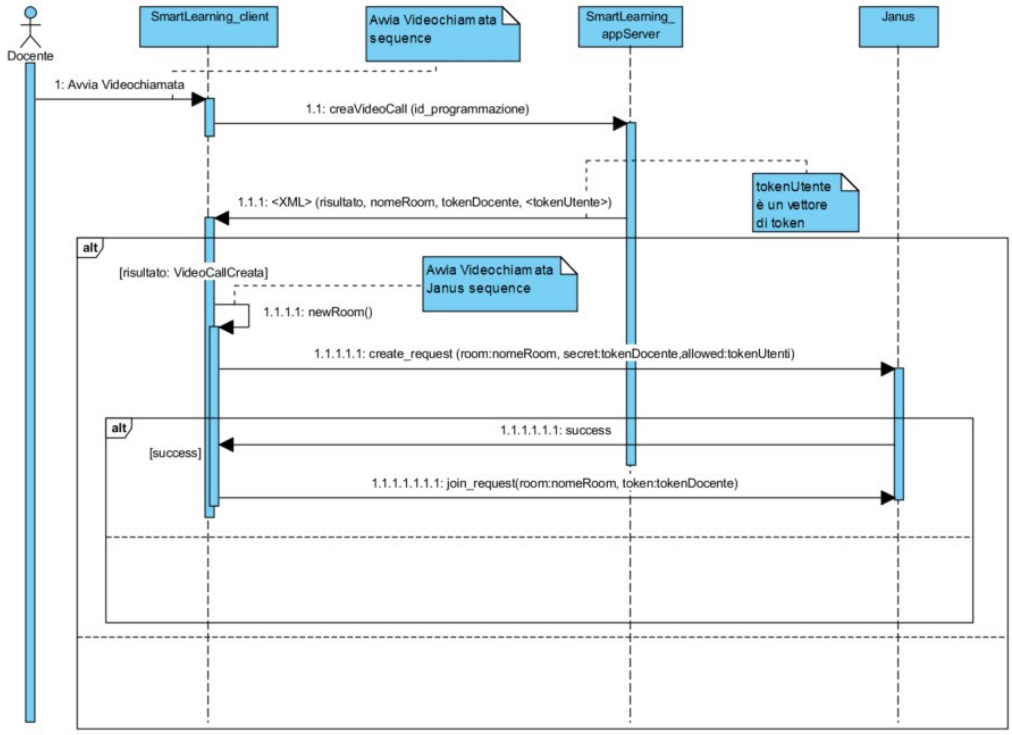

Figura 1: Sequence Diagram - Docente avvia videocall

In modo duale, uno studente per partecipare alla videocall:

- Egli contatta l'Application Server per ottenere il token a lui riservato per accedere alla Room;
- Lo Studente (Client Side) a quel punto si può interfacciare con Janus ed eseguire una semplice "Join" fornendo il proprio token.

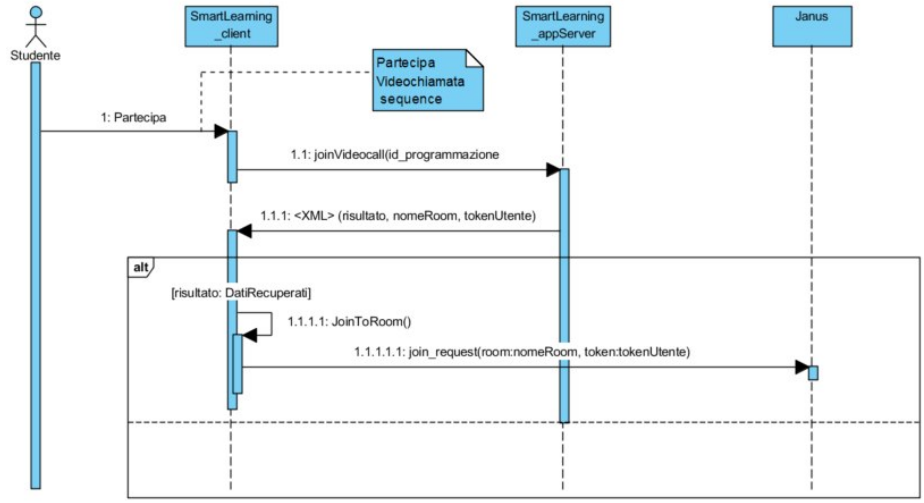

Figura 2: Sequence Diagram - Studente partecipa a videocall

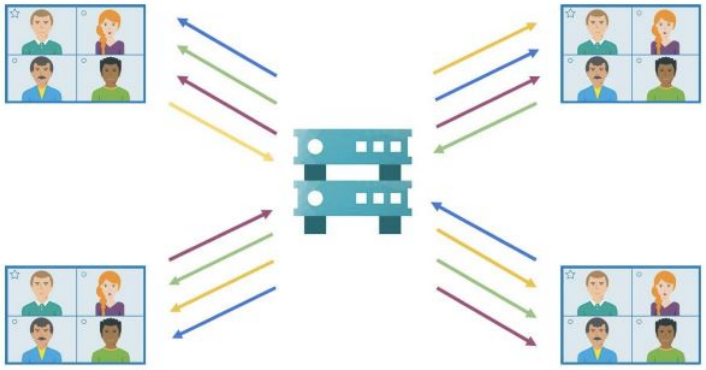

Figura 3: Selective Forwarding Unit

0.2.3 Mobile-Code (Code-on-demand)

L'architettura del sistema Smart Learning, nella corrente release, prevede quindi che l'interfacciamento con Janus sia Client-Side. Infatti, per ora, si sta assumendo che l'unico possibile client sia un browser, oramai sempre provvisto di engine sia per HTML, che per Javascript.

Di conseguenza, avendo il potenziale client dell'applicazione le risorse per poter eseguire il programma, è possibile delegargli l'interazione con Janus trasmettendogli sia la libreria "janus.js" che l'applicativo "mainpage.js".

Il vantaggio di questa scelta di design è quello di alleggerire il carico dell'Application Server, distribuendolo così sui vari client.

D'altra parte tale scelta può portare a problematiche di sicurezza: lo script javascript potrebbe essere malizioso, tuttavia l'engine javascript di cui sono dotati i browser è progettato per essere Client-Side e, di conseguenza, sono implementate delle policy di sicurezza (ad esempio che proteggono l'accesso al filesystem, alla videocamera, o al microfono) limitando le potenziali funzionalità del codice javascript.

0.3 Plugin VideoRoom

Come detto in precedenza, si è adottato il plugin "VideoRoom" di Janus, il quale fornisce l'implementazione di una SFU.

Una **SFU** (**Selective Forwarding Unit**) è un'entità in grado di ricevere più stream e decidere ognuno di questi a quali partecipanti deve essere reindirizzato. Uno dei principali vantaggi di questa architettura a confronto con una P2P è la scalabilità, infatti con un P2P ogni peer connesso caricherà flussi multimediali (audio, video, ecc.) N-1 volte (dove N è il numero di persone nella chiamata).

L'utilizzo di una SFU consente ai peer di caricare file multimediali e flussi di dati solo una volta. Il server quindi gestisce la distribuzione agli N-1 partecipanti. Questo essenzialmente aiuta a limitare la larghezza di banda consumata da tutti i partecipanti e consente di avere un numero maggiore di partecipanti alla chiamata.

La versione più aggiornata di Janus, inoltre, supporta il **multistream**, ovvero la possibilità di aggiungere più track dello stesso tipo ad una singola peerConnection. Nel nostro caso, ad esempio, è stato possibile gestire tutti gli stream in ricezione di un peer con una sola istanza PeerConnection.

L'utilizzo di una sola connessione presenta alcuni vantaggi, come la riduzione dei tempi di stabilimento della chiamata, poiché sono necessari meno Round di ICE(Interactive Connectivity Establishment – il protocollo per lo stabilimento della connessione tra browser) e un minor spreco di risorse lato client (comporta l'utilizzo di un minor numero di porte). Tuttavia questo approccio risulta essere più complesso nell'implementazione oltre che meno flessibile nell'aggiungere e rimuovere dinamicamente partecipanti.

0.3.1 Publish - Subscribe

Lo script "**mainpage.js**" che interagisce con Janus può istanziare fino a tre handler.

In generale, un handler instaura un canale bidirezionale di controllo e di notifica con Janus, permettendo al server Janus di fare la "push" di eventi verso i client, oltre a generare un canale dati unidirezionale (protocollo RTP) su cui viaggeranno i dati multimediali.

Nello specifico, due handler (publishers) sono entrambi dedicati a pubblicare un insieme di track che possono essere l'audio e il video della webcam o il video dello schermo, mentre il terzo handler (subscriber) è dedicato a ricevere lo stream di tracks degli altri utenti.

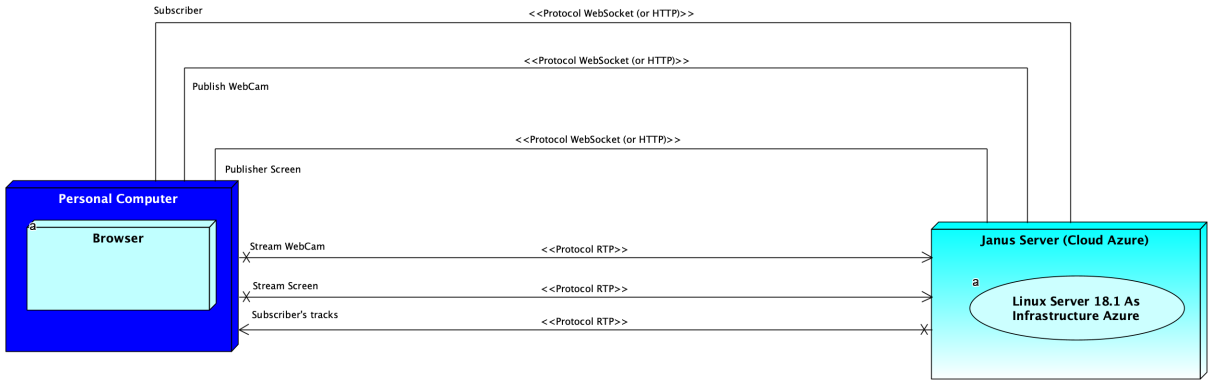

Figura 4: Deployment Diagram Client/Janus

Infatti, Janus adotta lo stile architetturale **Publish - Subscribe**, in particolare i topic a cui i subscribers possono sottoscriversi sono le singole track dei publishers mentre il Server Janus funge da intermediario.

Alcuni degli eventi generati da Janus verso i client sono:

- unpublished: comunica al subscriber che un publisher (a cui il subscriber era sottoscritto) ha smesso di pubblicare il proprio stream;
- publishers: comunica al subscriber che l'elenco delle tracks a cui è possibile sottoscriversi è stato aggiornato;
- leaving: comunica che un utente è uscito dalla room.

E' possibile gestire questi eventi tramite l'handler di sottoscrizione, il quale adopera un meccanismo di callback.

Per una maggiore documentazione sulla scelte implementative e, quindi, la definizione delle varie callback adoperate dai vari handler, si rimanda al codice e ai sequence diagrams.

Il vantaggio di avere Janus come intermediario è il disaccoppiamento dei client, esso è di tre diversi tipi:

- Spaziale: un client non ha bisogno di avere informazioni riguardo la raggiungibilità degli altri client;
- Temporale: i client possono essere volatili;
- Di sincronizzazione: non c'è bisogno che i client si sincronizzino, ciò evita numerosi blocchi.

D'altra parte, in una comunicazione multimediale tra N client il disaccoppiamento spaziale è indispensabile.

Lo svantaggio principale è il seguente: in questo modo Janus diventa un Single Point of Failure, ovvero nel caso di fallimento del server Janus la comunicazione tra i vari client non è più possibile.

0.4 Script di riposizionamento

Ogni utente può entrare/uscire dalla room, cosi come può condividere o meno lo schermo e/o la webcam. Ciascuna di queste azioni comporta una modifica nell'interfaccia degli stream. Di conseguenza, è stato implementato uno **script di ricalcolo e riposizionamento** dei video, il quale permette di avere in ogni momento una griglia ottimale rsipetto alle dimensioni dello schermo dal quale si effettua la chiamata.

```
function recalculateLayout() {
  Janus.log("RECALCUL");

  const aspectRatio = 16 / 9;

  const screenWidth = document.body.getBoundingClientRect().width;
  const screenHeight = (document.body.getBoundingClientRect().height);
  const videoCount1 = document.getElementsByClassName("Video-Stream").length;
  const videoCount2 = document.getElementsByClassName("Video-Stream hide").length;

  Janus.log("calcolo",videoCount1-videoCount2);

  const videoCount = (videoCount1-videoCount2);

  // or use this nice lib: https://github.com/fzembow/rect-scaler
  function calculateLayout(
    containerWidth,
    containerHeight,
    videoCount,
    aspectRatio,
  ) {
    let bestLayout = {
      area: 0,
      cols: 0,
      rows: 0,
      width: 0,
      height: 0
    };

    // brute-force search layout where video occupy the largest area of the container
    for (let cols = 1; cols <= videoCount; cols++) {
      const rows = Math.ceil(videoCount / cols);
      const hScale = containerWidth / (cols * aspectRatio);
      const vScale = containerHeight / rows;
      let width;
      let height;
      if (hScale <= vScale) {
        width = Math.floor(containerWidth / cols);
        height = Math.floor(width / aspectRatio);
      } else {
        height = Math.floor(containerHeight / rows);
        width = Math.floor(height * aspectRatio);
      }
      const area = width * height;
      if (area > bestLayout.area) {
        bestLayout = {
          area,
          width,
          height,
          rows,
          cols
        };
      }
    }
    return bestLayout;
  }

  const { width, height, cols } = calculateLayout(
    screenWidth,
    screenHeight,
```

```
61      videoCount,
62      aspectRatio
63    );
64
65
66      let root= document.documentElement;
67      root.style.setProperty("--width", width + "px");
68      root.style.setProperty("--height", height + "px");
69      root.style.setProperty("--cols", cols + "");
70  }
```