UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Colaboracion P2P para ambientes de desarrollo de usuario final basados en navegadores web.

AUTORES: Rodolfo Atilio Gonzalez

DIRECTOR: Dr. Sergio Firmenich.

CODIRECTOR: Dr. Gustavo Rossi.

ASESOR PROFESIONAL:

CARRERA: Lic. Sistemas.

Resumen

El browser, como lo conocemos, está evolucionando y la colaboración en la web, está tomando un papel central. Explorar la alternativa a la interacción de la arquitectura cliente-servidor por una arquitectura P2P permite que el usuario se conecte directamente con su otro par con la premisa de compartir contenido, descentralizar la información y aprovechar las tecnologías web. La propuesta es ofrecer Middleware P2P y un Framework para extender la funcionalidad del browser a través del uso de WebExtensions.

Palabras Clave

JavaScript, WebExtensions, WebRTC, HTML5, Arquitectura cliente-servidor, P2P.

Conclusiones

La capacidad de realizar colaboración P2P y motivar la descentralización nos acerca más a una Open Web Platform. Reducir los puntos centrales de información permite al usuario tener más de una alternativa para acceder a un servicio. Permitiendo que, sí dejara de funcionar, al menos lo pueda sortear, por medio de un usuario colaborador con el servicio.

Trabajos Realizados

Se desarrollo un Middleware y Framework con y para WebExtensions. Permitiendo al usuario la posiblidad de funcionar en modo P2P, motivando la colaboración en linea, desde el Browser. Permite al usuario construir su propia experiencia P2P con HTML5, JavaScript y otras librerias web.

Trabajos Futuros

Complementar y explorar el uso de una red Overlay sobre el Browser, por la cual se puedan maximizar el uso de distribución de archivos, base de datos distribuidas, privacidad de la navegación y estudiar la posibilidad de generar modelos y resolverlos mediante Machine learning P2P.

Fecha de la presentación: MES Y AÑO

Colaboración P2P para ambientes de desarrollo de usuario final basados en navegadores Web

Rodolfo Atilio Gonzalez

Universidad Nacional de La Plata

Facultad de Informática.

Directores
Director: Dr. Sergio Firmenich
Co-Director: Dr Gustavo Rossi

15 de junio de 2020

Índice general

Agradecimientos

Son muchas las personas que han estado en este camino recorrido. Por ello, quiero iniciar agradeciendo a mi familia, que ha estado presente y acompañandome en todo momento. También quiero darle un especial agradecimiento a mis directores de tesina, Sergio y Gustavo, que han sabido orientarme y guiarme en todo momento; en especial, por la paciencia que me han tenido y la capacidad de transmitir sus conocimientos. Sin más, muchas gracias a ellos.

Seguramente, me este olvidando de alguien más; por lo cual, agradezo a todos aquellas personas que saben y que han estado ahí para acomañarme cada momento bueno y dificil que ha transcurrido. A todos ellos, gracias.

Finalmente quiero a agradecer quienes lean este documento de tesina, por el tiempo y dedicación otorgada.

Resumen

La arquitectura cliente-servidor es la que utilizamos, en la actualidad, para acceder a sitios web desde nuestro navegador web. Con esta arquitectura, la web sigue creciendo y, a su vez, independientemente de ella, se incrementa el número de usuarios de internet. La masa de usuarios genera requerimientos de adaptación e integración de los servicios y contenidos existentes. Para satisfacer dichos requerimientos, es muy común que los usuarios extiendan el comportamiento de sus navegadores web con herramientas de Web Augmentation (WA[1]) y Mashups[2]. Que posibiliten satisfacerlos sin depender de los desarrolladores de las aplicaciones. Estas herramientas son denominadas, en la actualidad, WebExtensions[3].

Aunque los usuarios utilicen las técnicas como WA y Mashup integradas en WebExtensions, para mejorar su experiencia en la web, aún siguen dependiendo de los recursos de un servidor para, al menos, compartir y colaborar entre ellos. El simple hecho de compartir y colaborar entre los usuarios, a través del uso del browser, impacta en la interacción de la arquitectura cliente-servidor. Cambiar la interacción de la arquitectura cliente-servidor por una arquitectura P2P[4] permite que el usuario se conecte directamente con su otro par con la premisa de compartir contenido, descentralizar la información y aprovechar las tecnologías web.

El avance de dichas tecnologías hace posible que surjan nuevos proyectos que permitan, de alguna manera, dar forma a las premisas anteriormente mencionadas. Tal es así, el caso del nacimiento del proyecto BeakerBrowser[5]. Con BeakerBrowser es posible crear contenido web, como una simple página, y generar una url para luego compartirla. Todo ello, sin tener un servidor de intermediario.

[1] Web Augmentation: permiten modificar requerimientos funcionales y no funcionales de las aplicaciones sin depender de los desarrolladores

[2] Es una tecnica que permite generar nuevo contenido desde diferentes fuentes de información, y como una ventaja que su uso no requiere poseer un skil de programación.

[3] Es un artefacto de software que permite extender el comportamiento y funciones del browser.Link.

[4] Peer to Peer, conexión entre pares.

[5] Browser experimental, con soporte de tecnología P2P, Link pagina web, un navegador Web P2P experimental

Las nuevas tecnologías y la existencia de proyectos como BeakerBrowser, incentivan la colaboración y permiten una arquitectura P2P, planteando nuevas alternativas. Utilizando el auge de las WebExtensions en los navegadores, se plantea buscar y transformar los mecanismos de colaboración existentes, en pos de buscar alternativas a la necesidad de disponer de un servidor centralizado. Es decir, otorgarle al usuario una herramienta que le permita elevar la experiencia de espacio común de la información hacia otros usuarios de manera colaborativa, basada en P2P.

Este trabajo propone una arquitectura descentralizada e híbrida. Basada en navegadores web que permitan pensar, imaginar y desarrollar una capa de aplicación complementaria y colaborativa, tomando como punto de partida la web existente. Para ello, se propone el diseño e implementación de un Middleware[6] con la capacidad de transformar las WebExtensions. Añadiendo un medio de comunicación que permita convertir dicha WebExtension en una herramienta colaborativa mediante P2P, que pueda recibir peticiones y generar respuestas hacia otros usuarios que lo requieran y viceversa. Uno de los roles es de cliente-servidor que, a partir de un middleware P2P, otorga nuevas posibilidades de uso para las extensiones de usuario del navegador. Convertiendo sus datos en recursos (contenido web) y/o brindar servicios que puedan realizar operaciones con los datos de los usuarios remotos.

Durante el transcurso de este trabajo se abordaron la construcción, modelo de funcionamiento y las bases para realizar la conectividad P2P entre usuarios, a través de navegadores como así también, sus limitaciones. En cuanto a la construcción del mismo, presenta comunicación en tiempo real, funcionamiento independiente de la plataforma e independiente al medio de acceso de la red.

[6] Es una capa de software entre el sistema operativo y la aplicación encargado de ocultar las funcionalidades esenciales de comunicación

Capítulo 1

Motivación

Cuando estamos conectados a internet y visitamos un sitio, utilizamos un navegador web e interactuamos con el contenido de la página. Estos contenidos y servicios son consumidos cotidianamente por una gran cantidad de usuarios y, por ello técnicas como Mashups (para requerimientos de integración de información) y Web Augmentation (que permiten modificar requerimientos funcionales y no funcionales de las aplicaciones sin depender de los desarrolladores) han surgido como solución a los requerimientos de usuarios finales. A su vez, *estas son potenciadas por el uso de WebExtensions* [1], modificando el comportamiento del navegador y páginas web que el usuario navega, de forma tal que experimenta una mejora en la navegación web.

Las WebExtensions están implementadas en base a tecnologías client-side: HTML, Javascript y CSS; que han evolucionando desde 1990. Las mismas funcionan con recursos locales (CPU,memoria, almacenamiento), limitando la capacidad de compartir dichos recursos, de forma transparente entre los usuarios, a través del navegador web. Si el propósito de una WebExtensions requiere compartir recursos, es condición necesaria tener al menos un componente de Backend[1], que le permita el acceso a los usuarios remotos. A continuación, se mencionan alternativas existentes, que son utilizadas para obtener contenido web y mejorar la experiencia de navegación mediante colaboración:

[1] Es una abstracción que se encarga de gestionar las responsabilidades relacionadas con el funcionamiento de la arquitectura como puede ser el acceso al modelo, a la persistencia y recursos del servidor. Forma parte de uno de los estratos de separación de tres capas de la aplicación.

- Consumir un recurso web, a partir de APIs como servicio: el mismo principio de arquitectura cliente-servidor. Sin embargo, agrega comportamiento que permite consultar o realizar una acción en más de un servidor al mismo tiempo de forma transparente al usuario. Por ejemplo, el uso de una API del clima; el seguimiento de un paquete (producto de compra) a través de la web de mensajería del correo encargado de realizar el envío; visualizar películas por medio de streaming; entre otras.

- Consumir contenido web, a partir de una extensión de navegador: se instala una extensión con la capacidad de proveer funcionalidades de servicios que permiten manipular objetos web que, luego, se convierten en recursos para el usuario. Por ejemplo, el caso de Greasemonkey[2] y otras herramientas que permiten aplicar Web Augmentation y Crowdsourcing[3].

- Consumir y compartir recursos a través de un sistema de comunicación online: este tipo de aplicación permiten compartir mensajes, archivos, audio y video de manera distribuida y centralizada; también pueden ser P2P. Sin embargo, no todas proveen la capacidad de convertir recursos web en valor agregado para el usuario. Por ejemplo, podemos contar con una aplicación de comunicación online con varios usuarios pero, para gestionar el recurso propiamente de la web, necesitamos una herramienta complementaria que provea un mecanismo de abstracción, que pueda convertir el recurso de la web en un objeto de valor para otro usuario. Un ejemplo de ello sería descargar solo las noticias relevantes de un diario particular y, luego, compartirlas.

- Compartir almacenamiento y cómputo: el almacenamiento es un recurso finito y deseado. Por lo tanto, existe la posibilidad de alojar contenido en el Browser cliente en formato de JSON[4], que permita almacenar información por medio de P2P. En cuanto al cómputo, existe la posibilidad de enviar una cadena de datos a otro Browser que tengan más capacidad de cómputo, a fin de que puedan procesar los datos

[2] Es un manager de script, que se ejecuta en el browser del usuario. Permite instalar y utilizar una gran variedad de código JavaScript que elaboran los usuarios y, luego, lo suben al repositorio. Por ejemplo, personalizan Youtube, Vimeo, Wikipedia, entre otras. Link greasespot.

[3] en estes contexto, se tiene en cuenta la colaboración de tareas a partir de la división del trabajo en linea.

[4] JavaScript Object Notation, es un estandard abierto para el intercambio de datos en formato de objetos.

recibidos y retornar los resultados. Este trabajo se podría realizar en un proceso en segundo plano, de forma transparente al usuario, lo que dependerá de la carga de trabajo.

Para dar otro ejemplo, se analiza el caso de la WebExtension Diigo[5]. Diigo convierte al navegador web del usuario en un gestor de información personal, que permite gestionar marcadores de páginas web, notas y documentos. Todo el contenido de la información se guarda en servidores propietarios de la aplicación. Al día de la fecha, alberga un aproximado de nueve millones de usuarios, convirtiéndolo en uno de los servicios de información centralizado más destacado. Además de contar con una variedad de soportes para múltiples plataformas y compatible con la gran mayoría de los navegadores web. Para funcionar, requiere que el usuario inicie sesión en la página web de la aplicación, ya que la información se encuentra alojada en sus servidores.

En resumen, para construir una WebExtensions que funcione de forma colaborativa se requerirán, al menos: un componente client-side (la propia WebExtensions que altera el comportamiento del navegador) que se comunique con otro componente de backend (que permite gestionar a los usuarios que desean compartir entre sí). La capacidad de poder compartir y ofrecer recursos por medio de un navegador para generar una web abierta, colaborativa, que ayude a desvincular al servidor intermediario y ofrecer control de los datos a los usuario finales, se presenta como una posibilidad de explorar el comportamiento de un navegador web como P2P. Donde, no solo sea de acceso sino de servicio de recursos y aplicaciones, ya que las necesidades de un usuario pueden servir a otros. Las ventajas de poder operar en equipos por medio de una red compartida cambia la lógica de la arquitectura cliente-servidor hacia una arquitectura descentralizada. Generando un espacio para la reutilización de contenido y funcionalidades.

Por lo tanto, como respuesta a desvincular al servidor intermediario, han surgido implementaciones como Beakerbrowser. Un navegador Web que permite servir aplicaciones de manera descentralizada. Es decir, el valor agregado de Beakerbrowser es la arquitectura P2P, así como la capacidad de escalar aplicaciones web desde el browser del usuario. En definitiva, este trabajo se focaliza en complementar la arquitectura cliente-servidor en el ámbito de las WebExtensions, a fin de proveer un mecanismo de comunicación P2P.

[5] Diigo aplicación.

1.0.1. Objetivos y desarrollos propuestos

El auge de los servicios y contenidos en la web abre un sin fin de posibilidades para mejorar las experiencias de los usuarios. En una red P2P, podríamos generar y compartir contenido mediante técnicas como mashup y web augmentation (entre otras técnicas de enriquecimiento web) de forma descentralizada.

Se propone utilizar las WebExtensions de los navegadores para que se puedan comunicar en forma P2P. En base a esto, se plantea como objetivo general utilizar las tecnologías existentes, que se ejecutan en el lado cliente, para construir WebExtensions. Las cuales permitan ofrecer una arquitectura para funcionar en modo P2P y así poder complementarlas. Es decir, que dichas funcioncionades se puedan seguir realizando y, al mismo tiempo, ser compartidas como un servicio hacia otros. Como objetivos específicos se definen:

- Analizar y estudiar la tecnología existente que permite la conexión P2P por medio de WebExtensions.
- Analizar y estudiar la tecnología existente que permite la conexión P2P por medio de WebExtensions.
- Diseñar e implementar un Middleware que provee una capa de comunicación P2P.
- Diseñar e implementar un Framework que sea parte de la construcción de una WebExtensions para utilizar el Middleware y convertidas a P2P.

1.0.2. Estructura de la tesis

La finalidad de este apartado es dar a conocer la estructura de la tesis, teniendo en cuenta que, anteriormente, se ha puesto en manifiesto la motivación y los objetivos. A continuación se describen, brevemente, los capítulos:

- Capítulo 2: introduce al background, los mecanismos actuales y, de relevancia, para el trabajo presentado como así también, un foco en los mecanismos P2P para navegadores.
- Capítulo 3: introduce la tecnología base y casos de uso utilizados.
- Capítulo 4: visión general del enfoque, Middleware y Framework.

- Capítulo 5: Middleware y Framework implementación y funcionamiento.
- Capítulo 6: Construcción del ambiente de prueba y casos de uso.

Capítulo 2

Background

Las técnicas como Web Augmentation (WA) y Mashups han sido, son y serán, por excelencia, aquellas que han permitido a los usuarios obtener un conjunto de métodos capaces de proveer y generar contenido complementario, a través de diferentes fuentes de la web. Modificando y manipulando estilos, estructuras y comportamiento de las aplicaciones, así como se han podido reflejar en los trabajos, cita: [2], [3], [4] y [5]. Permitiendo así, extender la Web por medio de nuevos artefactos que aporten valor agregado a los usuarios, en un contexto de espacio común de la información.

2.1. Web Augmentation

WA es una técnica que permite agregar comportamiento y funcionalidades a las aplicaciones web, complementando la experiencia del usuario. Utiliza los recursos de la web que se encuentran disponibles en la nube o WebExtensions del navegador, ejecutados del lado cliente.

Con WA se permite cambiar el comportamiento de la web en tiempo real, a través de la modificación del DOM[1]. Esto permite personalizar la página web, según las necesidades del usuario. Se debe tener en cuenta que cada página tiene un sistema de renderización diferente y se necesita contar con determinadas reglas de funcionalidades, que se apliquen de lo particular a lo general.

[1] document object model, son las propiedades, métodos y eventos de los objetos que se generan en cada pagina web.Link reseña Firefox.

En general, existen repositorios de códigos disponibles en internet que pueden ser utilizados por los usuarios, permitiendo agregar comportamiento complementario a la web del día a día. Por ejemplo, un caso de uso frecuente son las páginas de repositorios de scripts elaborados por la comunidad de usuarios:

- https://www.greasespot.net/
- https://greasyfork.org/es
- https://openuserjs.org/

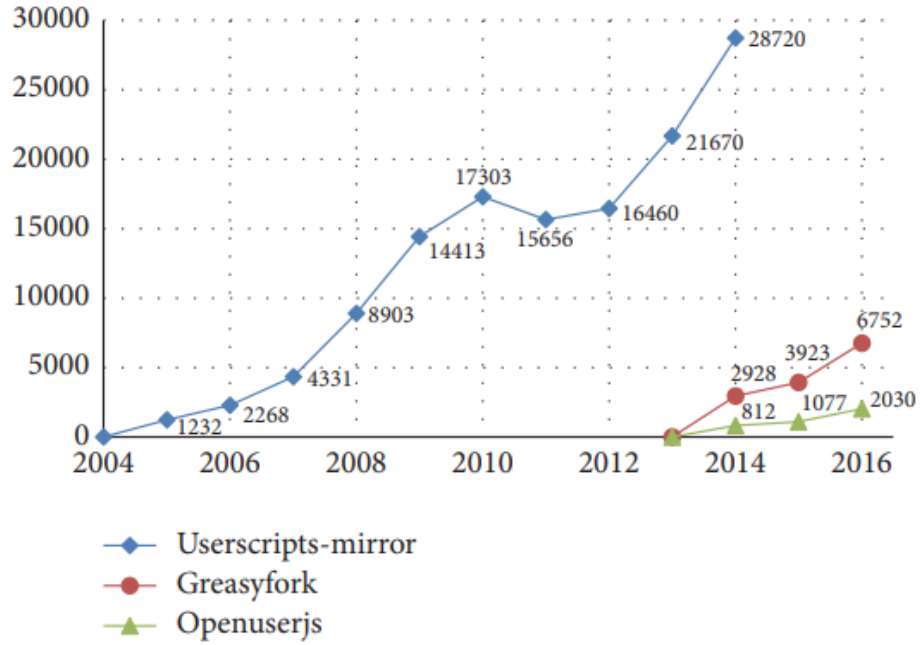
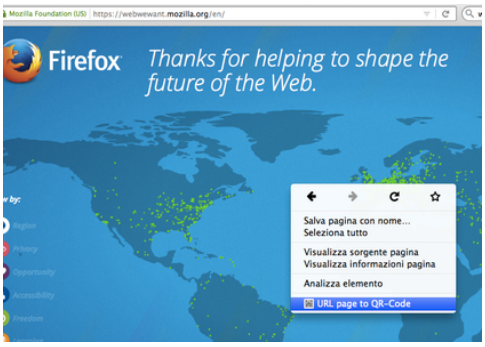


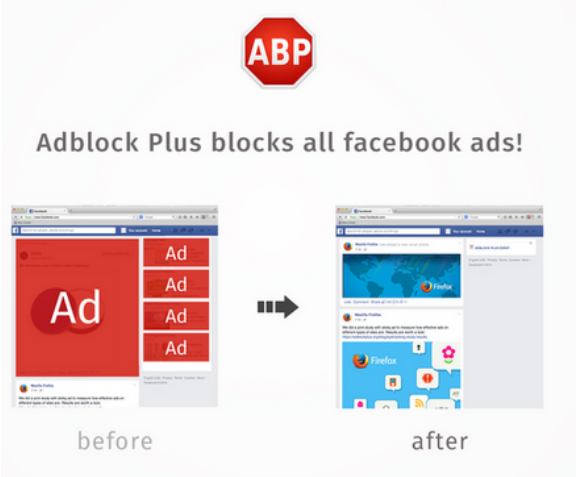
Figura 2.1: Crecimiento de actualizaciones de los repositorios [4].

A la lista anterior, se suman las WebExtensions de usuarios que proveen los navegadores web. Por lo tanto, el crecimiento de usuarios en la web y la disponibilidad de estas herramientas han impactado de tal manera, que se han convertido en una fuente de recursos de WA personalizada. Donde el usuario es el encargado de modelar y decorar la información obtenida a través

de codificación en JavaScript, siendo esta última la encargada de utilizarse por medio de la API del DOM o recursos externos como librerías de terceros, por medio de WebExtensions del navegador.



(a) WA botón derecho del mouse, interacción en el Browser.



(b) Bloqueo de publicidad durante la navegación del usuario.

Figura 2.2: Ejemplos combinados de WA en el Browser.

En las figuras 2.2, se puede observar una relación entre usuario-desarrollador-diseñador. Tanto el desarrollador como el diseñador, trabajan en conjunto, a fin de generar valor agregado a través de la web, con diferentes tecnologías y funcionalidades. En resumen, existe un usuario que puede anticipar un requerimiento funcional y no funcional, a fin de convertirlo en un recurso. Esto le permite poder realizar refactoring y personalización de WA (observer la

Figura 2.3: Tecnica de WA [4].

técnica WA 2.3) , ya que es posible utilizar distintos aumentadores y trabajar sobre ellos.

2.2. Web Mashups

Mashup, por sí mismo, no es una técnica propia de la informática sino que se puede encontrar en la economía, la gestión de empresas, el arte (dicha técnica de reunir distintos elementos en una sola unidad por definición artística es un collage [6]), entre otras.

En informática, proviene de conceptos de la Web 2.0 y el desarrollo web. El usuario, al navegar por las páginas web, se encuentra con la necesidad de un requerimiento que le permita combinar y/o fusionar los diferentes contenidos y servicios (los mismos pasan a convertirse en fuentes de información) en un solo lugar. Es decir, a partir de una aplicación web, es posible unificar los diferentes objetos en uno nuevo, facilitando la integración y disponibilidad para el usuario final. Al mismo tiempo, contribuir activamente en nuevas versiones de artefactos web, como así también de nuevas APIs. Las técnicas habilitadas para Mashup son:

- Software del lado del servidor.
- Software del lado cliente, en el browser del usuario.
- Una solución híbrida que contemple cliente-servidor.

Con las diferentes técnicas se pueden aplicar distintas tecnologías que permitan acceder a los datos y servicios para convertirlos en contenido útil para el usuario. Por ejemplo, en una WebExtensions de Browser, un Script desarrollado en el lenguaje JavaScript puede ejecutar un conjunto de subrutinas que personalizan la UI del usuario de forma parametrizables. Las

librerías y scripts que permiten funcionalidades se pueden descargar desde varios repositorios como, por ejemplo, aquellos que también fueron nombrados en WA 2.1.

Otro aspecto a tener en cuenta es el ciclo de vida de los Mashup. Al ser dependiente de un servicio, como puede ser una API, la cual se conecta a una página web (a uno o varios servidores o a un conjunto de otras API), la elaboración del contenido va a depender de la disponibilidad de la información, el protocolo de comunicación y del tipo de resultado que retorna. Ya que una actualización del retorno del contenido puede afectar a un conjunto de subrutinas que han sido elaborados para trabajar, con un determinado tipo de datos. Por lo tanto, si nuestra rutina funcionaba con un XML[2] y, luego, se realiza un cambio a un JSON, existe la posibilidad que el Mashup pueda fallar. Lo mismo puede suceder si un servicio de API cambia los parámetros de consulta. Otra dato a tener en cuenta es que, si el Mashup se ejecuta del lado servidor, se reduce la escalabilidad debido a la manipulación de datos.

A partir de Mashup se construye nuevo contenido, por medio de diferentes servicios y protocolos. Los servicios que consume una aplicación de Mashup se pueden producir a través de múltiples tecnologías web.

En resumen, WA permite complementar Mashup, a partir del valor agregado de la interfaz del usuario, generado por medio de artefactos como las WebExtensions. Cada complemento del navegador utilizado para Mashup permite agregar nuevas características en el navegador web y mejorar la integración con los sitios web.

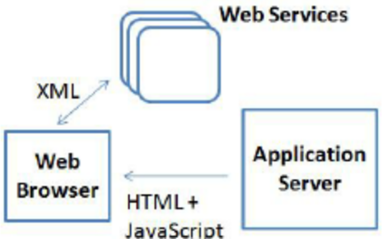


Figura 2.4: Una posible arquitectura de Mashup [7].

[2]Extensible Markup Language, lenguaje de programación de marcas

2.3. WebExtensions en el browser

WebExtensions nos permite extender y modificar el comportamiento del navegador. Generar una herramienta para el usuario que se construye a partir de la programación de un conjunto de archivos, escrito en lenguaje de programación JavaScript y un archivo de configuración en formato JSON. Como así también la evolución del browser como un contenedor de aplicaciones, en consecuencia a las WebExtensions. Ya que permite tener múltiples de ellas sobre el mismo browser y, en cada uno de ellas, se puede utilizar un gran número de APIs[3], que no solo permiten interactuar con el browser sino con el contenido que este genera.

Hay que tener en cuenta que no hay una compatibilidad 100 % que permita utilizar una WebExtension de un Browser a otro (cross-browser). Sin embargo, si se desarrolla en un estándar de codificación, puede ser retrocompatible. Por lo tanto, es el desarrollador de la aplicación quien otorga la facultad de proveer la compatibilidad de uso. Mientras más fiel a las APIs y ECMA-262[4] sea, más probabilidades existe de lograr compatibilidad entre browsers. Cabe destacar que, al menos, una vez por año se realizan cambios relacionados al ECMA, los cuales son adaptados por cada browser, en forma particular.

La relación de la comunicación de una WebExtensions con las aplicaciones web permite personalizar, no solo la renderización de la información sino agregar nuevas funcionalidades. En la comunicación hay que tener en cuenta el acceso al DOM de las páginas y la capacidad de establecer al menos, un canal de comunicación, según el tipo de contexto que se utilice. Cuando de contexto se trata, hay que tener en cuenta: background y content-scripts. Los contextos permiten a una WebExtensions interactuar con datos y eventos de las páginas web. Cada contexto favorece a extender las funcionalidades del browser y mejorar la experiencia del usuario con las herramientas que esten utilizando como parte de las WebExtensions.

Con la construcción de WebExtensions en el browser no solo tenemos acceso a una arquitectura cliente-servidor, sino que es posible generar una arquitectura híbrida de conexión, a partir del uso de APIs, que son provistas

[3] Interfaz de programacion de aplicaciones, Link acceso WebExtensions API

[4] hace referencia al código numérico de estándar, es una especificación de uso y mejoras en cuanto al lenguaje Javascript. Aunque no significa que una vez publicado el standard este esté disponible, ya que depende de cada browser.

por las tecnologías disponibles dentro del motor de cada browser. Una de las ventajas que presenta, es la capacidad de poder integrar técnicas aumentación web, a partir de artefactos ya construidos. Otra ventaja es que, son desarrolladas en un lenguaje de programación como JavaScript, que permiten una integración más accesible con el backend del servidor (Javascript también permite funcionar como backend por medio de NodeJS[5]).

En cuanto a la compatibilidad de tecnologías, en la actualidad existen empaquetadores, client-side de aplicaciones web. Con la capacidad de organizar y ensamblar una cantidad de librerías, que permiten generar funcionalidades a partir de un solo archivo. A partir de estos empaquetadores, es posible realizar la tarea de integración de forma más sencilla, ordenada y portable. Está ventaja permite a las WebExtensions tener otro potencial dentro de un Browser, ayudandolos a agregar funcionalidad a partir de librerías. Algunos suelen ser aplicaciones como WebPack[6] y Parcel[7].

Otra de las posibilidades es agregar funcionalidades, a partir de los repositorios de scripts disponibles para los usuarios. A partir de ellos, construir más herramientas de aumentación web que han demostrado ser de utilidad y validas para hacer frente a la demanda de personalización de aplicaciones [8]. Finalmente, en lo que respecta a seguridad, cada WebExtensions notifica al usuario qué tipo de permisos va a utilizar sobre el browser del usuario, para funcionar. Hay que tener en cuenta que las políticas son particulares de cada WebExtensions. A continuación, se describe las especificaciones de configuración de una WebExtensions.

[5]Es un entorno de ejecución multiplatorma que permite ejecutar código JavaScript fuera del navegador web. Esta diseñado para escalar.NodeJS, Licencia, tracemark.

[6]Es una herramienta a partir de la cual se genera un artefacto de software compuesto por diferentes componentes, facilitando la estructura de depdencias y encapsulandolo.Link pagina.

[7]Similar a WebPack es una herramienta que permite generar un paquete de aplicacion web, con la diferencia que no requiere configuracion.Link pagina.

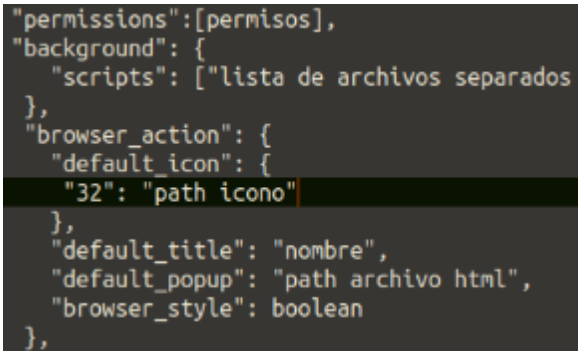
2.3.1. Configuración de Manifest en una WebExtension

```
"manifest_version": version,
"name": "nombre",
"version": "n.y",
"description": "una descripción",
"icons": {
  "48": "path icono",
```

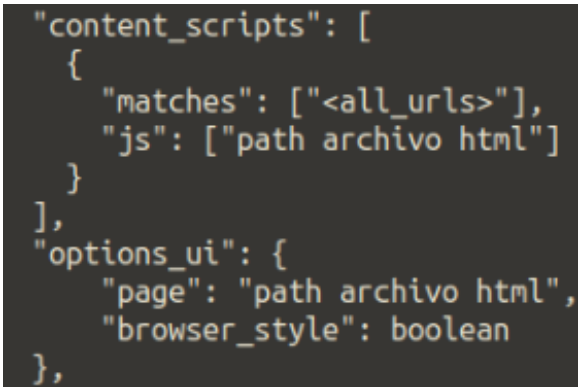
Figura 2.5: Configuración de manifest en formato JSON.

En la figura 2.5, se puede observar el archivo de configuración que posee manifest_version, el cual se declara en formato numérico (en este proyecto se utiliza la versión 2). El nombre y la descripción son un string; la versión es otro, string con la diferencia que posee un formato major.minor.release[8]. La sentencia icons requiere un tamaño de icono con su respectivo path completo.

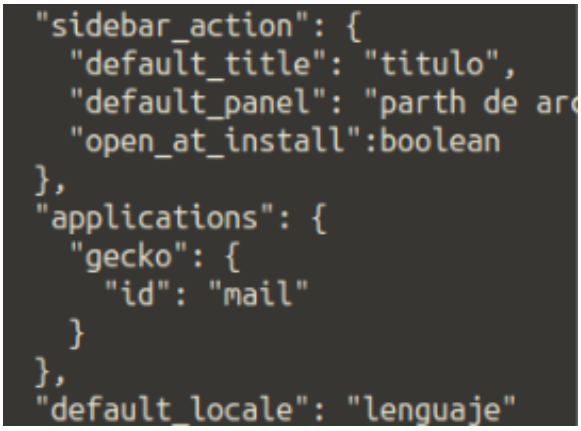
[8] Acceso al tipo de versión web.



(a) Permisos.



(b) Configuración de content-scripts y options-ui.



(c) Configuración de sidebar_action y application geck-id.

Figura 2.6: Ejemplos configuración de Manifest.

En las figuras 2.6 podemos observar la sentencia permissions[9]. Aquí se integran la lista de privilegios separados con coma, ellas son necesarias para que funcionen correctamente las WebExtensions. Estos privilegios se muestran al usuario cuando se instala. Es el usuario quien decide aceptar los permisos de seguridad y continuar con la instalación. En el background2.6, se pide agregar una lista de archivos separados con coma, con sus respectivas rutas de localización.

En browser_action 2.6 se realiza el mismo procedimiento que icons. En la sentencia default popup, se puede observar que posee una referencia hacia la página principal, la cual se presenta en formato HTML. La misma se presentará al usuario al realizar click en el correspondiente icono de la barra de herramientas del browser. En cuanto al content-script[10], su configuración es similar al background. La diferencia en este contexto es que se activa el archivo JavaScript, que interactúa con las páginas que se cargan en los tabs del browser del usuario. En esta configuración, aparece la sentencia *<all_urls>*, que significa que estará ejecutándose en todas las páginas que visite el usuario mientras utiliza el browser. En *options_ui* solo tenemos que agregar el path correspondiente al archivo html, que proveerá al usuario las opciones de configuración de la aplicación.

En esta última figura2.6 podemos observar el sidebar_action, que nos permite editar el titulo y agregar un valor boolean true o false. Según el valor, se visualizará durante la instalación o no y solicita el path completo del archivo HTML, donde reside la página a renderizar. En applications se solicita un id[11]; este es un identificador como, por ejemplo, una dirección de mail. El id es importante, ya que identifica a la aplicación y, a partir de ella, podemos comunicarnos desde otras WebExtensions.

Un vez que la aplicación se instala, se genera un id único, en forma aleatoria, es cual es utilizado por el browser. La sentencia *default_locale* define el lenguaje. Esto posibilita agregar traducciones con archivos de diccionario de datos. De está forma, permite tener una aplicación multilenguaje.

Hay que tener en cuenta que si existe una falla en una línea de código o una falla por error interno de codificación de alguna dependencia de la

[9] Configuracion de permisos de la WebExtension.Link versi'on web.
[10] Seccion encargada de proporcionar la manipulaci'on de las paginas web.
[11] Identificador que proporciona acceso unico a la WebExtensions.Link web.

aplicación, el browser la detecta e impide que este funcione correctamente, alertando al usuario que se presentó un error durante la instalación. Es necesario prestar atención a las utilidades que nos proveen los browsers para realizar la carga de las WebExtension en modo debug.

Figura 2.7: Pantalla en Firefox cuando se presenta una WebExtension en modo testing.

En la figura2.7, se puede observar la leyenda *extensiones temporales*, ya que no se encuentran firmadas ni empaquetadas, sino que es un modalidad de debug. La modalidad *debugging* permite al usuario interactuar con la aplicación; cargándola directamente desde manifest.json, permitiendo realizar una trazabilidad de carga y funcionamiento. Para acceder a este modo debug en Mozilla Firefox se debe ingresar a la barra de direcciones y escribir about:debugging. Luego, cargar el archivo de manifest correspondiente.

En el proyecto de desarrollo de la aplicación se utiliza esta metodología, a fin de agilizar todo el proyecto, ya que no existen diferencias de uso ni limitaciones. En la figura2.8 se puede observar la relación del *manifest* con todos los tipos de scripts. Cada script estará restringido a los permisos que el desarrollador aplique y se ejecutará en un contexto separado del browser.

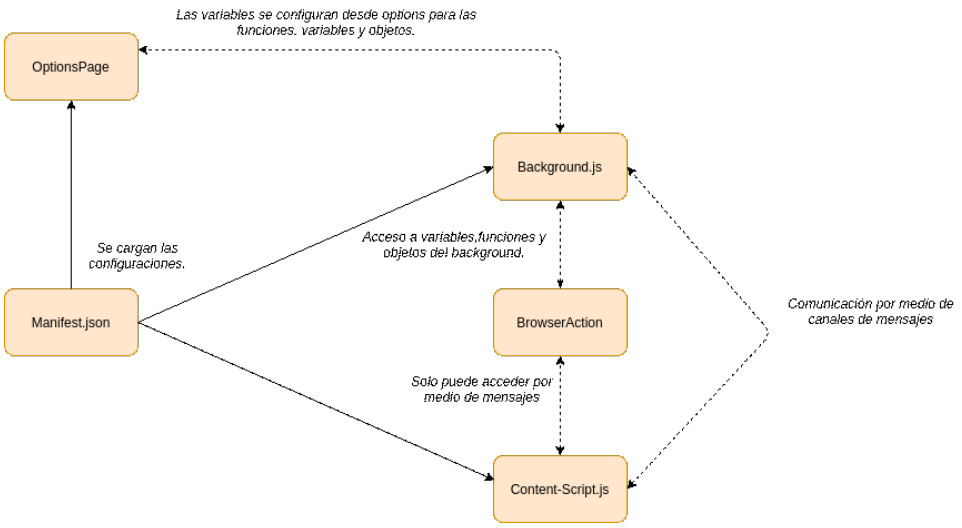


Figura 2.8: Componentes de una WebExtensions.

2.4. P2P y Web Browsers

Una web basada en P2P permitiría generar un nuevo ecosistema descentralizado de servicios web, tanto para WA como Mashup. Y, a su vez, poder complementar el uso de la arquitectura cliente-servidor.

Mientras, la web 2.0 ya había cambiado la forma de crear y obtener contenido web para los usuarios, por medio de los blogs, wikis, servicios web y API de aplicaciones, aún se encontraba en desarrollo y crecimiento. Sin embargo, tiene las ventajas de mejorar la interacción, el contenido dinámico y la mejora de coloración en línea. Este otorgo a su mayor activo: el usuario como creador de contenido.

La integración de la web semántica vino acompañada de una gran demanda de usuarios. En 2016, con el auge del soporte de los browsers en los dispositivos moviles, se estimaba más de un billón de usuarios con oportunidad de acceso a internet y nuevas tecnologías, que permiten crear nuevas demandas a través de dispositivos inteligentes [9]. No esta lejos pensar que todos ellos colaboren e interactúen entre sí. A traves de la nube, los multi-dispositivos, la colaboración globalizada y el surgimiento de la IA(Inteligencia Artificial); con la premisa de que la red se adaptará al usuario y, todos juntos construyan y mejoren la web semántica.

Con las mejoras en tecnologías web como HTML5, WebRTC[12] y WebExtensions, el usuario puede desarrollar addons/extensiones de navegadores, para interactuar con la sesión y al mismo tiempo, otorgarle la posibilidad de realizar cambios en relación al servicio que esté consumiendo en la web desde su navegador y compartirlo con otros usuarios. Permitiendole al usuario tener una web colaborativa.

Teniendo en cuenta lo expuesto anteriormente, el proyecto plantea la posibilidad de focalizar la contrucción del uso de la comunicación P2P entre navegadores. Inspirado en la idea browsers de borde. Sin embargo, en lugar de sólo alojar aplicaciones web para llegar a una Web descentralizada, se pretende utilizar la funcionalidad P2P, a fin de poder servir para aplicaciones por parte de los usuarios finales, a través del uso de los contenidos y servicios existentes de la Web. Por lo tanto, los artefactos construidos podrán comunicarse directamente entre sí. Esto implica que los artefactos puedan funcionar como server y cliente, y dependiendo de la aplicación, podría compartir y colaborar en determinado momento [10].

[12] Es una tecnologia de comunicación en tiempo real, que puede ser utilizada mediante una API de browser.

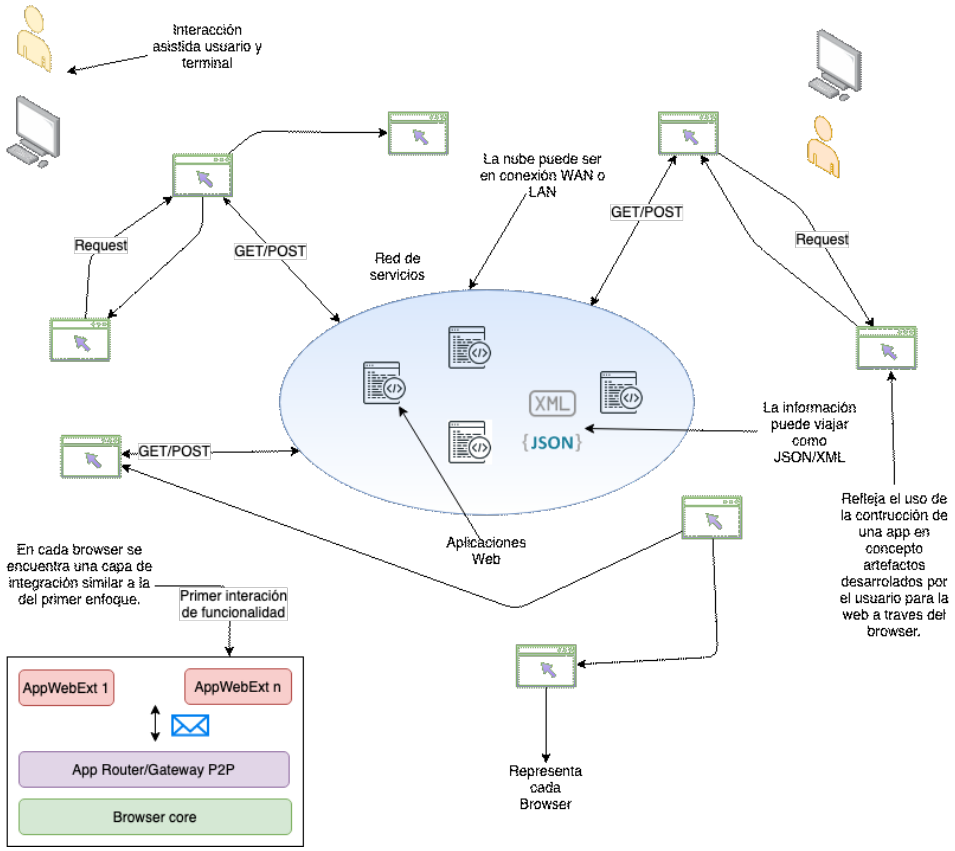


Figura 2.9: Primer aproximación del proyecto.

En la figura2.9 se observa la arquitectura de funcionamiento, en modo de colaboración entre browsers. Donde los usuarios habilitan determinadas aplicaciones, según las funcionalidades desarrolladas por parte de sus artefactos, que se encuentran extendiendo el comportamiento de los browsers. Cada artefacto puede responder peticiones o enviar solicitudes, según el rol de funcionamiento.

2.4.1. Resumen

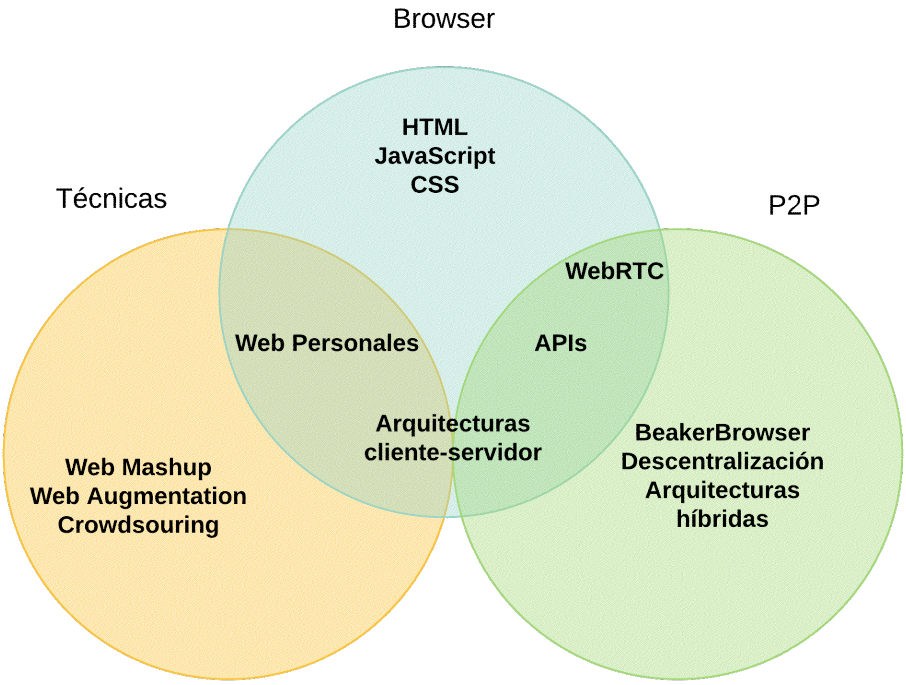


Figura 2.10: Relación entre las tecnologias y proyecto.

En la figura2.10 vemos cómo las tecnologías, técnicas y P2P están relacionadas y, a partir de ellas, es posible re-utilizarlas. Por ejemplo, en la generación de artefactos que se integren en los navegadores. Como se puede notar, existe una colaboración entre las páginas personales, la arquitecturas cliente-servidor y la capacidad de utilizar descentralización a partir de tecnologías como WebRTC y APIs.

A partir de la figura anterior 2.10, se puede concluir que el browser, con sus tecnologías disponibles, puede funcionar como una plataforma de aplicaciones [9] porque:

- El browser está en control del usuario, le permite gestionar políticas de uso en webextensions y conexiones de red.
- Es independiente del dispositivo.

- Un browser puede ser cliente y servidor por medio de websockets, WebRTC y javascript.
- Permite una integración a nivel protocolos (WebSocket, HTTP y entre otros) y otorga control sobre ellos.
- Permite realizar una mayor integración de hardware como pueden ser IOT con Web of things y cloud integration (servicios y herramientas que consumen tecnologías a través de Restful API).
- Permite trabajar con aplicaciones web, por medio de tecnologías distribuible como son las WebExtensions.
- Permite reutilizar código gracias a JavaScript porque funciona en el cliente como en el servidor.

Capítulo 3

WebRTC: Tecnología de base y trabajos relacionados

3.1. Resumen

En este capítulo se realiza un recorrido por la tecnología base y la relación en el trabajo, así como los diferentes proyectos de relevancia que han sido de aporte al mismo

3.2. Historia

El protocolo WebRTC fue liberado en 2011, a través de Google. Luego, fue estandarizado por la IETF (Internet Engineering Task Force) y W3C[1]. Con el tiempo, fue soportado como una API de browsers denominada WebRTC API, que se encuentra disponible en la mayoría de los navegadores web y funciona sobre los dispositivos(todos aquellos con browsers compatibibles con WebRTC), permitiendo acceder a los periféricos de entrada como WebCam y micrófono.

El potencial de esta tecnología, presente en los browsers desde el 2012 y, con ayuda de HTML5, permite construir y pensar una red colaborativa a través del uso de la API de WebRTC. En la medida en que el ancho de banda disponible se incrementó y los costos en hardware, como procesadores, memoria y almacenamiento, bajaron; las barreras de implementación en tecnología interactiva, disminuyeron. A su vez, para el 2018 se estimaba un

[1] [2]

total de 1.5b de smartphones en ventas (basado en los artículos [11] y [12]) y se estimaba un potencial, el mismo año, de 4.7b de dispositivos (se tiene en cuenta la investigación del artículo [13]) con soporte para WebRTC.

Por lo tanto, el auge en dispositivos conectados a internet que se encuentran con soporte en WebRTC integradas en los browsers son suficientes como llamar nuestra atención. Pudiendo esperar un horizonte bastante prometedor en la capacidad de las próximas aplicaciones web (la llegada de HTML5, mejoras en Javascript , Apis de Browsers y WebExtensions) con capacidad de integrarse a una red descentralizada por medio de los browsers.

3.3. Arquitectura y funcionamiento en WebRTC

WebRTC es un protocolo de comunicación en tiempo real, que ofrece una arquitectura multicapa capaz de ensamblar una arquitectura P2P. Entre las capacidades que ofrece son:

- Colaboración multiusuario a través de los navegadores web.
- Comunicación síncrona a través de la red en una topología estrella[3].
- Eficiencia en la gestión de ancho de banda en la transmisión de video a través del soporte de hardware como GPU[4]. Permite utilizar compresión de video.
- Permite comunicación P2P.
- Permite utilizar diferentes protocolos de transporte.
- Permite conectarse en una LAN y WAN y funciona a través de un firewall por medio de NAT[5].
- Es compatible con la mayoría de los browsers de forma nativa.

Una de las desventajas principales es que requiere construir una infraestructura de soporte que tenga en cuenta:

[3]en está topología todos los nodos se encuentran concentrados en un solo punto de conexión. Cada nodo se comunica a con el resto a través del punto común de convergencia.

[4]A graphics processing unit, es un procesador graco.

[5]Network Address Translation RFC 1918, permite que una red interna pueda comunicarse a internet a través de una dirección pública enmascarando la conexión de forma estática o dinamica.

- Definir un protocolo de comunicación para un señalizador.
- Crear un servidor señalizador.
- Crear un servicio de NAT.
- No provee soporte para service worker y procesos en background.
- Finita cantidad de conexiones.
- Ocional:
  - HA.
  - Escalabilidad.
  - Balanceo de carga

El funcionamiento en los navegadores no es simplemente instanciar un objeto que represente al funcionamiento de la API de WebRTC y realizar llamadas sobre ella. Sino que tiene una serie de requerimientos que hay que tener en cuenta, pero gracias al soporte de los navegadores y el estandar, permite que su utilización resulte de forma simple y directa. Todas las funciones necesarias requieren al menos tener en cuenta lo siguiente:

- Un protocolo de comunicación.
- Una api integrada del browser que pueda funcionar con Javascript y el servicio en backend, que permite la comunicación entre dos browsers.

Como WebRTC es un conjunto de varias funcionalidades, en nuestro caso, para el desarrollo se consideró un alcance base, el cual se resume a continuación.

3.3.1. Alcance y funcionalidades de WebRTC API

Como se mencionó al inicio de la sección inicial de WebRTC, la comunicación no es sencilla, por lo tanto, necesitamos entender qué tipo de objetos son y qué funciones son las que necesitamos utilizar. WebRTC API se formó pensando en los tres conceptos fundamentales: comunicación entre peers, envío de datos entre ellos y la transmisión audio/ video. Siguiendo los conceptos bases, se han tenido en cuenta las siguientes interfaces:

RTCPeerConnection: esta interfaz es la responsable de permitir la conexión local y remota entre peers del browsers. Este es el encargado de utilizar la infraestructura de comunicación mencionada en la arquitectura, a fin de administrar los eventos, estados y sesiones de los peers en los browsers. Los protocolos que utiliza durante la conexión son SDP[6] y ICE[7]. Ellos son indispensables a la hora de utilizar el servicio de señalización. Por su puesto, durante su creación, se puedan tener en cuenta la configuración sobre qué tipo de conexión se realizará. Es decir, si es en una LAN o si es por medio de NAT; en ese caso, incluiría una lista de servicios de STUN/TURN, que permitirá conectar los browsers desde conexiones que puedan facilitar el pasaje de los mensajes por medio de firewalls. Para ello, se facilita la configuración en formato JSON. De esta interfaz nos interesa utilizar los siguientes métodos(se tienen en cuenta sólo los métodos necesarios para el proyecto de investigación aplicada de tesina. Existen otros métodos, los cuales no son utilizados en este proyecto.):

- CreateDataChannel: para permitir el envío bidireccional de datos.
- Onclose: handler de cierre de conexión.
- Ondatachannel: handler de apertura de conexión del canal de comunicación. Utilizado para gestionar eventos sobre el canal creado.
- Oniceconnectionstatechange: permite saber el estado de conectividad entre peers.
- Onicecandidate: permite obtener información de la red del peer remoto y, en dicho evento, transferir los datos del peer local hacia el remoto.
- CreateOffer: permite crear una petición de oferta de conectividad hacia el peer remoto. Está creación genera un SDP.
- CreateAnswer: permite crear una respuesta hacia la oferta recibida del peer remoto. La respuesta genera un SDP.
- SetRemoteDescription: permite guardar información relacionada con la sesión del peer remoto.
- Close: cierra la conexión del peer.

[6] session description protocol. Utilizado por medio de JSEP.

[7] Interactive Connectivity Establishment, es un protocolo de intercambio que permite establecer la sesión sobre NAT. [14].

- AddIceCandidate: guarda información del objeto que se genera durante la negociación con el peer. Esto permite saber información de direccionamiento IP y puerto de conectividad. Está información estará relacionada con la red, si es directa, o por medio de STUN/TURN[8].

RTCSessionDescription: esta interfaz permite obtener información de la potencial negociación soportada entre peers, es decir, las opciones en las cuales se pondrán de acuerdo cuando negocien. Es lo que intercambian en cada oferta y respuesta. Este contiene información del protocolo SDP que sigue el RFC3264[11], que será soportado durante la configuración de la sesión.

RTCIceCandidate: esta interfaz es el encargado de obtener los detalles del tipo de conectividad a realizar, es decir, a partir de qué y cómo se conectara entre los peers. Tiene información relacionada con la conectividad de red, dirección IP, puerto, prioridad y protocolo utilizado para la conexión. Sigue el RFC5425[12]. Esto permite que cada peer pueda conocer la potencial topología de red sobre la cual se van a conectar y con cuál ruta de red lo harán.

RTCDataChannels: esta intefaz es la encargada de realizar un canal bidireccional de comunicación de datos entre los peers. Este puede funcionar con SCTP, sigue el RFC4960[13] y permite una comunicación fiable[14] y no fiable, así como las propiedades para enviar los datos en orden o fuera de orden, por default es en orden. La implementación de los mismos se encuentran soportada por los browsers actuales. Dichas propiedades no pueden ser cambiadas una vez que el canal se encuentra establecido. Por lo que se debe tener en cuenta que establecer una comunicación no fiable requiere configurar los parámetros maxRetransmits y maxRetransmitTime; al no configurarlos, se genera una configuración fiable. Un dato importante a tener en cuenta es que el protocolo SCTP es independientes del stack de WebRTC; además, de que posee las ventajas de los protocolos TCP y UDP. En cuanto al

[8] servicios de red para comunicación WebRTC a través de firewall. Session Traversal Utilities for NAT cumple RFC5389[9] y Traversal Using Relay NAT cumple RFC5766[10]

[11] Acceso al RFC.

[12] Acceso al RFC.

[13] Acceso al RFC..

[14] Hace referencia al tipo de conexión, una comunicación fiable es aquella que esta orientada a la conexión TCP.

tamaño de información, hay que tener en cuenta que es transparente al usuario, ya que el propio protocolo se encargará de utilizar las conexiones necesarias y las porciones de datos a enviar en una cantidad finita de paquetes de red, a través del protocolo especificado. A continuación, se presentan los tipos de datos soportados y las funciones de la interfaz que nos interesa utilizar:

- Tipos de datos: String, Blob, ArrayBuffer y ArrayBufferView
- Onopen: handler que se utiliza a la hora de abrir el canal.
- Onclose: handler que se utiliza a la hora de cerrar el canal.
- Close: método utilizado para cerrar el canal.
- Send: método utilizado para enviar datos.

3.3.2. Arquitectura

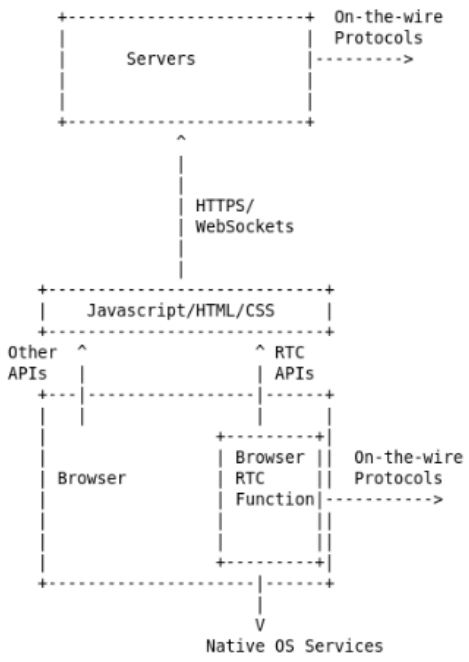


Figura 3.1: Arquitectura WebRTC basado en JSEP(JavaScript Session Establishment Protocol) [15].

En la figura 3.1 se pueden apreciar los ítems a tener en cuenta para obtener una comunicación satisfactoria. Esto representa un esquema de conexión necesario a implementar denominado *capa de infraestructura de comunicación centralizada* [15]. La misma permite el intercambio de información entre los browsers, a fin de realizar la conexión entre pares. Está capa es la que representa:

- El browser con la api WebRTC.
- Un servicio de DNS.
- La capa de tecnología HTML5 y Javascript más posibles librerías complementarias.
- El servidor encargado de encaminar la comunicación mediante WebSocket/XHR[15] con cada browser, haciendo el papel de señalizador.

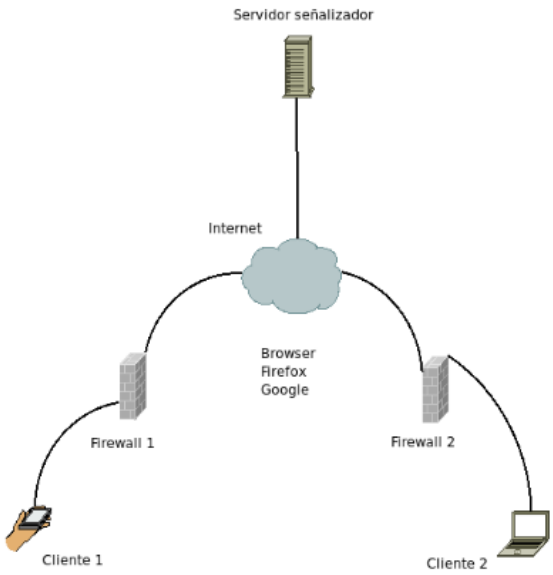


Figura 3.2: Arquitectura WebRTC con señalizador.

En la figura 3.2, vemos un escenario donde dos clientes se encuentran conectados a través de routers/firewall. Los Browsers, acceden a internet para

[15] XMLHttpRequest

conectarse hacia el servidor señalizador. Hay que tener en cuenta que, si nos conectamos por medio de internet, será necesario un servicio extra que ayude a ser visible la conexión de cada browser, ya que cada uno no necesariamente cuenta con su propia dirección ip pública, sino que puede pertenecer una red LAN local (puede ser red por cable o WIFI). Esto es así, ya que las direcciones ip públicas están limitadas, por lo que cada uno se encuentra conectado a internet por medio de NAT. En el caso de conectarse por medio de NAT, además del servidor señalizador, será necesario contar con un servicio que permite saltar las restricciones del router/firewall. Para este caso se utilizan servicios de STUN/TURN.

WebRTC necesita de una funcionalidad de señalización para que la conectividad exista entre dos browsers. La señalización se utiliza para mantener la compatibilidad con las tecnologías existentes y evitar la redundancia de implementación. Además de delegar la responsabilidad respecto de redes como latencia, perdida paquetes y reducir el costo operacional (nos permite abstraernos de la topología de red). Por lo tanto, se utiliza un estándar, que está definido para este propósito, basado en una arquitectura denominada JSEP. La cual permite desacoplar el establecimiento de las conexiones interactivas, guardando los estados de las sesiones en servidores externos al browser.

Figura 3.3: Esquema minimalista JSEP.

En la figura 3.3 se puede apreciar la función de la arquitectura JSEP, desacoplada del browser. La leyenda *App-specific Signaling* se encuentra en el lugar de la aplicación que implementan la funcionalidad de coordinación de señalización entre ambos browsers. Asumiendo una conectividad satisfactoria entre el servidor señalizador y el browser, se procedería de la siguiente forma:

1. El browser inicia un proceso para crear una oferta por medio del servidor señalizador.

2. El browser del otro extremo responde a la oferta a través del señalizador

e inicia un evento, a fin de guardar los datos de la oferta de conexión (guarda información de SDP).

3. El browser inicial recibe la respuesta e inicia el evento de guardar los datos (información de SDP) de la conexión.

4. Detrás de escena aparece el protocolo ICE, encargado de conectar los peers. Este utilizara, en caso de ser necesario, los servicios de STUN/-TURN. Será el encargado de enviar las direcciones IP y puertos para cada Browser. Este procedimiento ocurre de forma asíncrona. Durante este proceso, cada browser dispara un evento para enviar y guardar los datos que provee el protocolo.

5. Una vez establecida la conexión, se procede a su uso.

Hasta aquí, se exploran los pasos básicos para realizar la conexión entre dos browsers. Si bien WebRTC tiene un público aggiornado hacía audio y video, nuestro foco es la capacidad de uso de enviar flujo de datos (streams) a través de los Browsers, a fin de ser utilizado como medio de comunicación y pasaje de información entre aplicaciones. Para ello, se utiliza una tecnología soportada en los Browsers, denominada *DataChannel*. A partir de ella, es posible generar un canal de comunicación simétrico que permite comunicarse entre los browsers.

3.4. Usos de WebRTC

A continuación se mencionan proyectos que fueron construidos a partir de tecnologías como: Javascript, HTML5, WebSocket, WebRTC y otras librerías externas en JavaScript. En los siguientes casos de uso, la colaboración entre pares es el núcleo principal de cada trabajo. Poniendo de manifiesto diferentes tipos de servicios.

Como tipo de servicio se refiere a casos de uso como mejorar el tráfico de la red para stream, mejorar la construcción de modelos tridimensionales, construir un servicio utilizando al browser como plataforma de acceso y hasta mejorar la capacidad del browser, a fin de construir aplicaciones sobre este.

3.4.1. USE Together

Es un desarrollo basado en WebRTC, para permite colaborar con proyectos CAD a través de los navegadores de los usuarios y, según el paper [16]

permite explotar la funcionalidad de las placas de video, a fin de compartir los recursos y mejorar la capacidad de realizar modelos 3D.

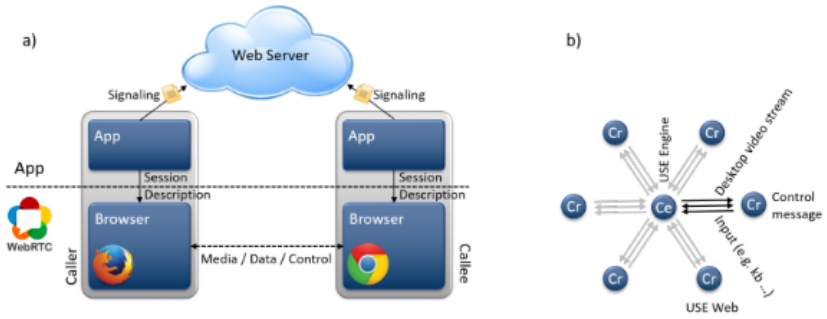


Figura 3.4: Arquitectura WebRTC USE together.

3.4.2. P2P multi-server basado en dash.js

En este proyecto se utiliza la capacidad de ancho de banda de los peers por medio de WebRTC y la arquitectura que ofrece Dash.js, permitiendo consumir menos ancho de banda. La capacidad de funcionar a través de una red NAT, facilita el funcionamiento de los peers a la hora conectarse. Su funcionalidad híbrida, según el paper [17], mejora las prestaciones ya que permite obtener contenido desde los servidores y desde los peers. Además, permite que cada peer adapte su propio algoritmo de calidad.

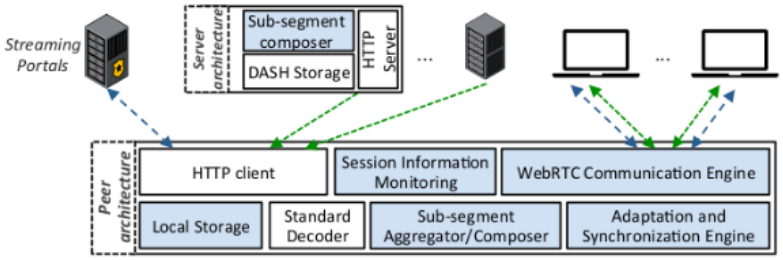


Figura 3.5: Arquitectura WebRTC P2P multi-server basado en dash.js.

3.4.3. Pando

Pando, a Volunteer Computing Platform for the Web [12], es una aplicación que permite utilizar recursos de cómputo por medio de una red P2P, a través de WebRTC. El core del proyecto es proveer a cada usuario con un browser que le permita tener la capacidad de ser voluntario, ofreciendo recursos de cómputo y procesando streams remotos. Por lo tanto, sería posible ejecutar una función sobre un stream en un peer para que, luego, este pueda enviarlo hacia otro permitiendo, por ejemplo, la ejecución encadenada de funciones y así sucesivamente sobre una red P2P.

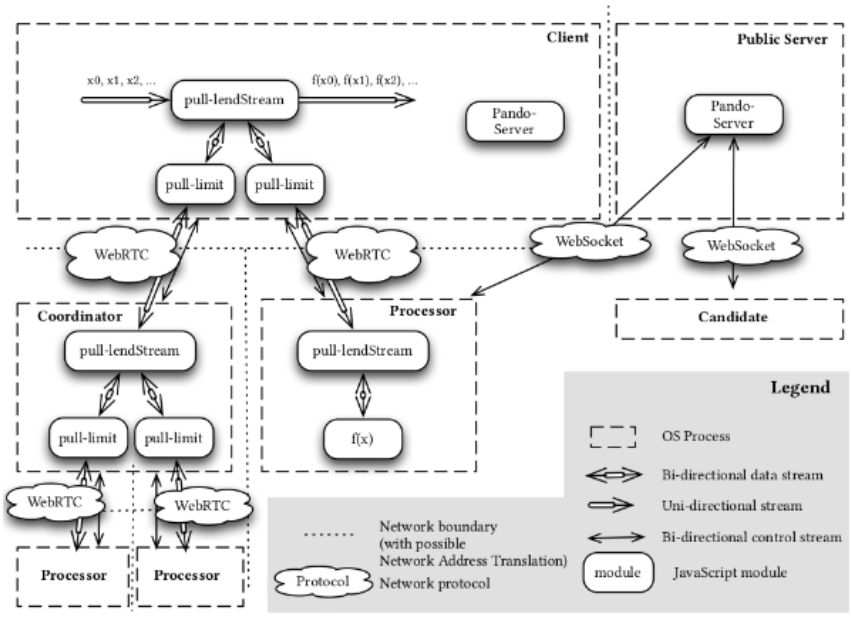


Figura 3.6: Arquitectura WebRTC Pando.

3.5. Otros enfoques P2P

En los casos anteriores se expuso cómo es posible utilizar el browser a través de tecnologías como WebRTC y lograr conectividad P2P, así como también, se ha mostrado la integración con los browsers con múltiples tecnologías colaborativas. Si bien existen otros enfoques que se están desarrollando y, por ende, se está modificando la forma de utilizar los browsers, a continuación exploramos un browser experimental P2P nativo y por otro lado,

una herramienta que permite integrar al browser como medio de recurso compartido.

3.5.1. Ngrok

Nos permite exponer servicios de forma pública, a través de la construcción de un túnel de red cifrado que permite utilizar el protocolo HTTP/HTTPS sobre este. Por lo tanto, es posible tener un servicio en localhost desde internet y utilizar sus recursos de nuestra aplicación web. La cual se encuentra funcionando localmente a través de una url pública, que otorga la herramienta de Ngrok.

Está aplicación habilita la posibilidad de observar los cambios en forma dinámica, permitiendo una arquitectura híbrida cliente-servidor. Por lo tanto, no necesariamente, otorga la posibilidad de convertir al browser en un servidor P2P de contenidos. Sin embargo, permite poder construir un puente de comunicación entre los browsers locales de cada usuario, por medio de un servicio externo, que puede estar ejecutando en cada usuario local. En resumen, Ngrok se convierte en un recurso compartido a través de la red y el browser.

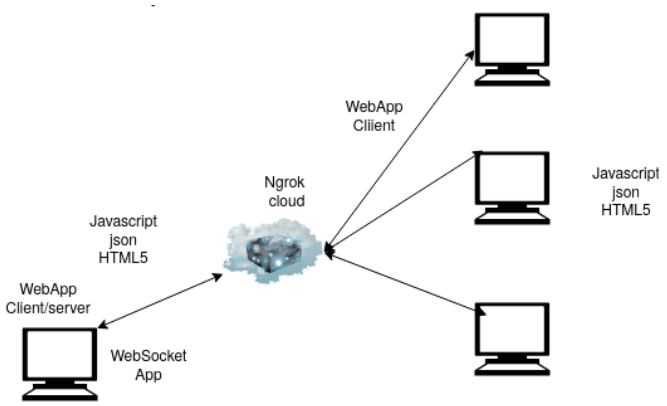


Figura 3.7: Arquitectura de caso de uso Ngrok.

3.5.2. BeakerBrowser

BeakerBrowser es una caso de uso que tiene, como foco principal, convertir el browser convencional en un browser P2P. Es el proyecto de un browser

experimental que permite navegar por la web como los navegadores convencionales. Como ventaja, ofrece la capacidad de ser una fuente de aplicaciones en red, permitiendo compartir y generar contenido P2P, todo a través del navegador web.

Con BeakerBrowser es posible generar un sitio web y compartir archivos. A continuación, se mencionan algunos atributos de este proyecto:

- Utiliza el protocolo Dat[16].
- Utiliza APIs para la construcción de sitios.
- Permite editar la configuración del sitio creado en tiempo real.
- No utilizan servidores como intermediarios. Existe la posibilidad subir el contenido (una copia del sitio), de forma tal de hacerlo visible a través de una url.

Cada aplicación creada con BeakerBrowser permite utilizar las funcionalidades P2P. Por ejemplo, Twitter permite al usuario publicar mensajes en su muro virtual, permitiendo los comentarios de otros usuarios. Lo cual requiere de una arquitectura cliente-servidor, teniendo en cuenta un stack de frontend-backend. Pero, por otro lado, la misma funcionalidad en BeakerBrowser sólo requiere que el usuario genere un sitio con HTML, JavaScript y CSS donde, luego de publicarlo y compartirlo con otros usuarios, el servicio ya se encuentra activo en el navegador del cliente, permitiendo la interacción de mensajes con otros usuario. Todo esto, sin utilizar servidores, ya que puede ser ejecutado desde cualquier PC o notebook.

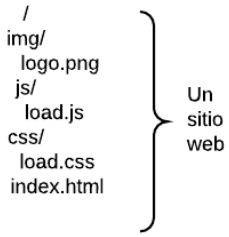


Figura 3.8: Sitio sober BeakerBrowser.

[16]Es un protocolo que permite funcionar de forma similar a Http con la diferencia que permite descentralizar los accesos y Es un sincronizar grandes cantidades de archivos [18]. Link descripciíon oficial.

En la figura 3.8 podemos observar un ejemplo de la estructura de un sitio web. Solo con este contenido, permite convertir el sitio convencional en P2P, una vez que esté funcionando en BeakerBrowser, a través del protocolo Dat.

Funcionamiento BeakerBrowser

BeakerBrowser utiliza el protocolo Dat para navegar por una web P2P y una interface DatArchive para generar contenido. Dat es un protocolo diseñado para compartir datos entre máquinas y funciona con y sin internet. Los desarrolladores del protocolo dejan al alcance del usuario final una herramienta disponible que mejora la experiencia de usuario y su facilidad de uso.

Dat permite y contribuye a la construcción de la próxima generación de la web. Toma la experiencia, el uso y algunas partes de tecnología de las herramientas como Git[17], BitTorrent[18] y Dropbox[19] para su diseño e implementación.

La diferencia entre la dirección url convencional con Dat es que utiliza una key (un conjunto de string de longitud finita) como: *dat://key/recurso*, donde key es un conjunto de caracteres (64 en total en formato hex encode, basado en ed25519 [19])) de 32 bytes. Está *key*, representa una llave pública, con ella, permite ser accesible a otros usuarios y verificar los datos. Mientras que el recurso es un directorio o archivo accesible por medio del protocolo Dat.

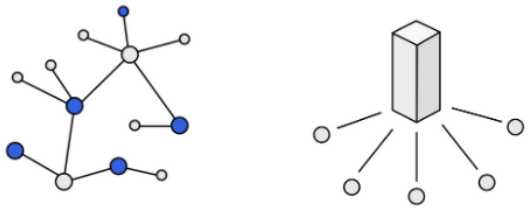


Figura 3.9: Topologias de conexión.

[17] Es un programa que permite acceder a un sistema de control de versiones.
[18] Es un protocolo, para la carga y descarga de archivos P2P.
[19] Es un programa que con arquitectura cliente-servidor que permite utilizar la nube como un espacio de almacenamiento

En la figura 3.9, vemos dos topologías de conexión de nodos. La primera, es la que refleja a BeakerBrowser mientras que con la segunda, vemos la web centralizada arquitectura cliente-servidor.

Desde el punto de vista de las tecnologías disponibles, se puede afirmar que existen intentos positivos en relación a una web colaborativa, que complemente la experiencia del usuario. El Browser, como lo conocemos, transmite e interactúa con los documentos, aplicando un conjunto de tecnologías que facilitan la renderización de contenido y consumo de recursos de los servidores en la red.

Si tenemos presente, tecnologías que nos permitan poder expandir los límites de los browsers y, al mismo tiempo, aprovechar las mejoras que ellos proveen; entonces, se puede afirmar que WebRTC, JavaScript y las WebExtensions pueden formar parte de una conversión complementaria para los browsers, con la capacidad de expandir su funcionalidad cliente-servidor.

3.6. Conclusión

No debemos olvidar que WebRTC no solo es una tecnología disponible en los Browsers, sino que existen fuera de ellos también, lo que significa que existe más de una forma de conectarse. Teniendo en cuenta que todos los Browsers son endpoint (puntos finales), es posible implementar aplicaciones en otros lenguajes de programación que utilicen librerías que imiten el comportamiento de los Browsers, ya sea para ser endpoint, gateway, SFU (selective forwarding unit) o, simplemente, ofrecer servicios a través de la red. Algunos lenguajes de programación con soporte WebRTC:

- JavaScript.
- C++.
- Java.
- Python.
- GO.

Teniendo en cuenta las tecnologías disponibles de cada lenguaje de programación y las librerías que soportan, en cuanto a las conectividades de redes, las aplicaciones de uso pueden ser varias. Por ejemplo, se puede implementar un servicio web que se ejecute en un servidor que esté conectado

a un Peer browser y, mediante el envío de mensajes, poder comunicarse para ofrecer los recursos y realizar un forward hacia otros peers.

Su capacidad no está limitada a una red LAN sino que puede funcionar con topologías que se encuentren fuera de la LAN como internet. La topología de la red se puede ver beneficiada para aquellos usuarios en los casos donde la latencia de red es mayor o tienen restricciones de acceso a la red.

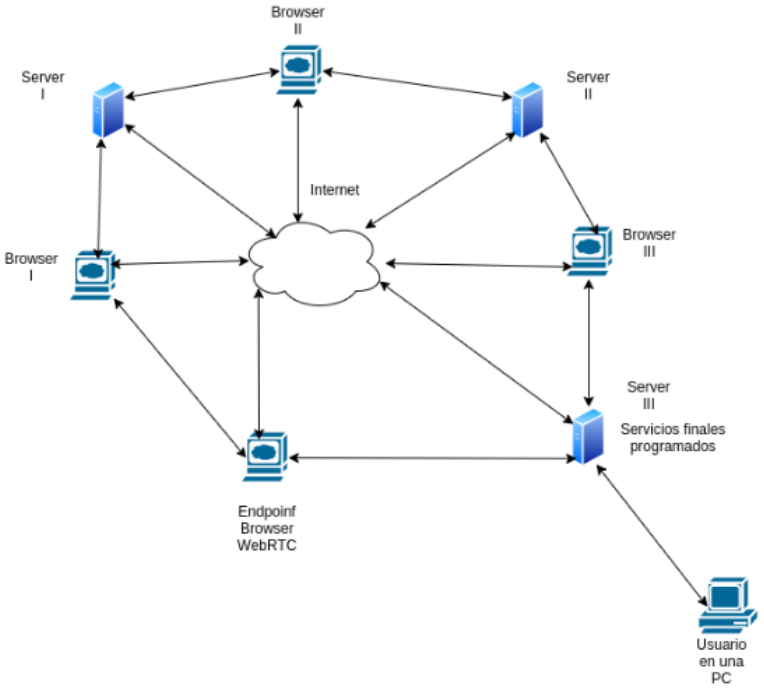


Figura 3.10: Caso de uso topología de conexión y gateway en modo borde con Browser.

Como podemos ver en la figura 3.10, el uso de P2P en los Browsers tiene un horizonte prometedor y más allá de ellos, también. Sin ir más lejos, la existencia del proyecto como el caso BeakerBrowser desarrollado en NodeJS, aunque sea experimental, es un paso alternativo de navegación y colaboración en red que permitirá otorgar al usuario un rol principal en la nube.

Después de todo, también es una realidad que la propagación de conec-

tividad en ancho de banda, en los dispositivos finales, como puede ser el browser del usuario final, se está volviendo más diverso. Y ante esta diversidad, la API de WebRTC, está a nuestro alcance para explorar desde el punto de vista de la colaboración en línea. En nuestro caso, el proyecto no estará basado en audio/video sino que intenta complementar el uso de las WebExtensions como puente de conexión entre el browser y las aplicaciones (teniendo en cuenta la tendencia en el uso de browser de borde como en paper [20]) que funcionan como WebExtension, permitiendo complementar la experiencia del usuario en línea, a partir del comportamiento y tecnologías existentes en la web. Este proyecto de desarrollo, permite exponer un medio de comunicación entre pares sobre una extensión del navegador. Siendo está una pasarela de comunicación entre el usuario y la funcionalidad concreta. Está funcionalidad permite un posible servicio de valor agregado en cada browser de usuario.

Capítulo 4

Visión general del enfoque

4.1. Browser P2P a partir de Middleware y Framework

Disponer de un browser P2P requiere poder resolver tareas más allá de las convencionales. Es decir, además de las tareas y funcionalidades que ofrece el browser de manera transparente al usuario, requerirá tener en cuenta agregar, al menos, una funcionalidad de comunicación que no afecte ni cambie la lógica de uso, a la cual el usuario ya se encuentra aggiornado. Por lo tanto, se necesita una capa de abstracción con la capacidad de proveer un mecanismo de comunicación P2P, que permite al browser poder desligarse de la complejidad de comunicación.

En conclusión, nace la necesidad de la construcción de un Middleware que se encargue de controlar las operaciones y comunicaciones P2P. Y en cuanto a la necesidad de facilitar el desarrollo de WebExtensions personalizadas, que puedan utilizar la modalidad P2P, se complementa la necesidad de abstracción a través del uso de un Framework, que permite desligar la complejidad de su implementación y solo preocuparse en el requerimiento necesario para el desarrollo de una WebExtension clásica.

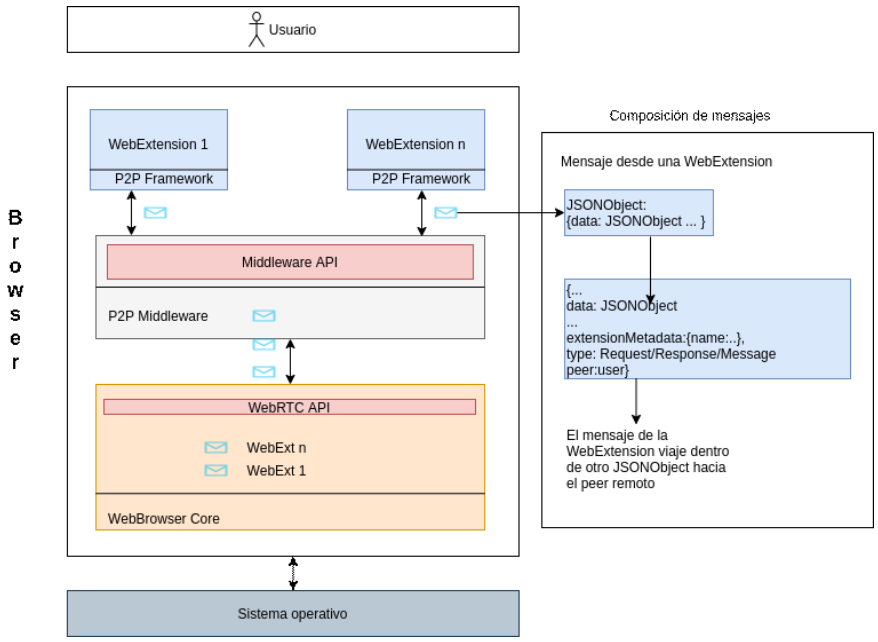


Figura 4.1: Capas del proyecto.

En la figura 4.1, se puede observar la arquitectura de funcionamiento completa que corresponde al funcionamiento emparejado del Middleware con la WebExtensions y su Framework [21].

4.2. Middleware P2P

En pos de unificar las capacidades fundamentales de un entorno de comunicación P2P, se diseño y desarrollo un Middleware con WebExtensions que permite a otras WebExtensions, comunicarse con este y que se abstraiga de la complejidad de la capa de comunicación.

Permitiendo así el desarrollo de aplicaciones P2P, de forma que el usuario se pueda abstraer de los detalles de las tecnologías de comunicación y solo preocuparse por el envío de mensajes a través de él. Se construyó en base a un conjunto de herramientas, funciones y protocolos compatibles para poder comunicarse por medio de mensajería en formato JSON.

El Middleware complementa la experiencia del usuario mientras se en-

cuentre en línea. Al construirlo sobre un browser, estamos utilizando un componente de software de la web y al desarrollarlo con WebExtensions, lo integra como parte de la Open Web Platform [9]. Su construcción está basada en el conjunto de herramientas con tecnologías libres y disponibles de la web.

Con la idea de extender la conectividad, más allá de la arquitectura cliente-servidor, y complementar la colaboración del usuario por medio de WebExtensions, se realizó una implementación que permita al usuario ser parte de una red P2P, por medio del Browser, desde una WebExtension.

Para ello, se requiere tener las siguientes consideraciones:

- Generar una infraestructura de comunicación que permita funcionar a través del browser.
- Tener, al menos, una API de browser que permita establecer la comunicación independientemente del medio y que permite ser representado como un usuario.
- Permitir el envío de información entre los peers y gestionar los mensajes.

Los tres ítems anteriores han sido tenidos en cuenta para lograr conectividad P2P a través del Browser, por medio de WebExtensions. Razón por la cual, se implementó un Middleware que tenga en cuenta la arquitectura JSEP con WebRTC, la tecnología HTML5 y JavaScript.

Una de las ventajas de la implementación de un Middleware es que no solo sea para mensajes P2P sino que se pueda extender el concepto de mensajes hacia un canal de comunicación software con software y software con usuario.

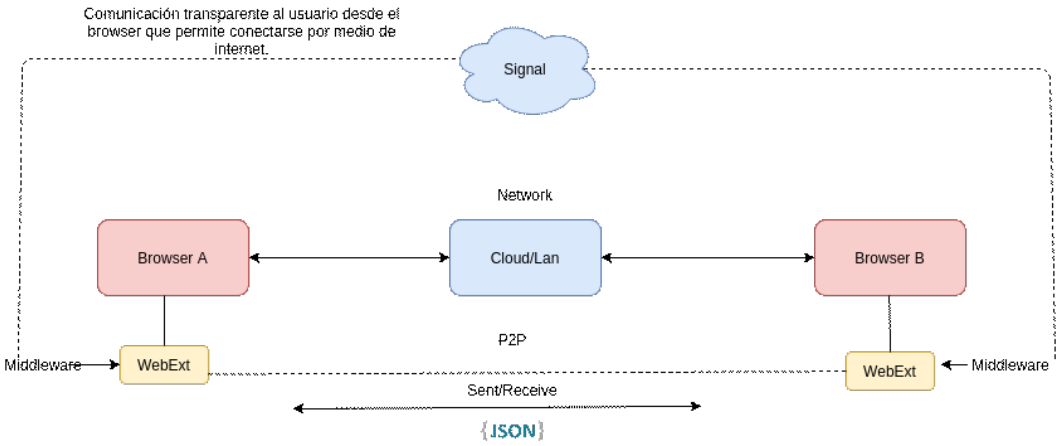


Figura 4.2: Modelo minimalista del Middleware.

En la figura 4.2 se representa la funcionalidad de comunicación entre dos Browsers, por medio del Middleware. Por una lado, ambos Browsers deben tener la WebExtension instalada y, a partir de allí, poder comunicarse. Durante la implementación se trató, al menos, de conseguir lo siguientes ítems:

- Que las comunicaciones sean transparentes.
- Que la comunicación funcione de forma asincrónica.
- Si se desarrolla una WebExtensions personalizada, que pueda utilizar la funcionalidad de P2P de la manera más transparente posible.
- De ser posible, que pueda escalar.

La comunicación, al ser basada en una arquitectura JSEP con WebRTC, requiere tener en cuenta el rol de la parte backend (servicio de señalización) y la tecnología en el browser cliente. Por lo tanto, se implementa la comunicación de la arquitectura JSEP, por medio de WebSocket, y la tecnología en la capa de comunicación del cliente, por medio de WebRTC, nativa del browser.

Al haber varias capas de comunicación, se implementó un modelo de objetos que permite hacer uso de la funcionalidad de mensajes y de la interacción del usuario con la WebExtension.

4.3. Framework para el desarrollo de extensiones

Cuando hablamos de comunicación software con software, hablamos de una modalidad que genera valor agregado al Middleware P2P porque permite a nuevas WebExtension poder utilizar la modalidad P2P, a través de él. Entonces, cuando un usuario desea utilizar la comunicación en modo P2P en su WebExtension, va a necesitar un protocolo de comunicación entre extensiones.

En esta primera versión, se elaboró un medio de comunicación que permite comunicarse a través de un Framework que se debe agregar en cada WebExtension personalizada, a fin de poder utilizarlo. En un primer intento, se optó por utilizarlo mediante un link de descarga. Así como toda librería en Javascript, luego de ser descargada en el browser, es posible crear instancias de lo que sea que está contenga. Sin embargo, replicar el comportamiento de objetos desde una página web, en el caso de uso necesario, puede llevar a que se genere y destruya tantas veces como la página se recargue o se corte y vuelva la conectividad de la red. En ambos casos, no es un escenario ideal.

El uso de herencia de WebExtensions se descarta ya que no es posible heredar desde otra WebExtension. Además de que cada WebExtension se encuentra en formato empaquetado y se ejecuta en contextos del Browser diferentes. Por lo tanto, generar una WebExtensions que funcione por medio de P2P, necesita aplicar un modelo de objetos que permita extender la funcionalidad y facilitar el comportamiento que cada una de ellas realice. En consecuencia, se utilizó la herencia de objetos en JavaScript como metodología y así distribuir en un archivo, para ser ejecutada en el contexto de background de la WebExtension. El uso de una clase abstracta permitirá ayudar a extender la funcionalidad y complementar una WebExtension de colaboración sobre otras extensiones, y, sobre todo, teniendo en cuenta al menos, tres funciones necesarias:

- Conocer a los peers.
- Recibir mensajes.
- Enviar mensajes.

Una instancia que herede de una clase abstracta permite enviar mensajes en formato JSON por medio de un canal de comunicación, que se genera entre las extensiones por medio de un identificador único entre ambos. A

partir de la comunicación entre el Framework y el Middleware, es posible enviar mensajes a los Peers. La WebExtension personalizada podra usar el Framework para:

- Conectar cada extensión particular a un canal de comunicación de la extensión P2P.
- Tener un método dentro de la extensión personalizada que envíe los mensajes a través del canal. De forma tal, que cuando llegue al Middleware P2P, lea el mensaje y lo envié a quien corresponda.

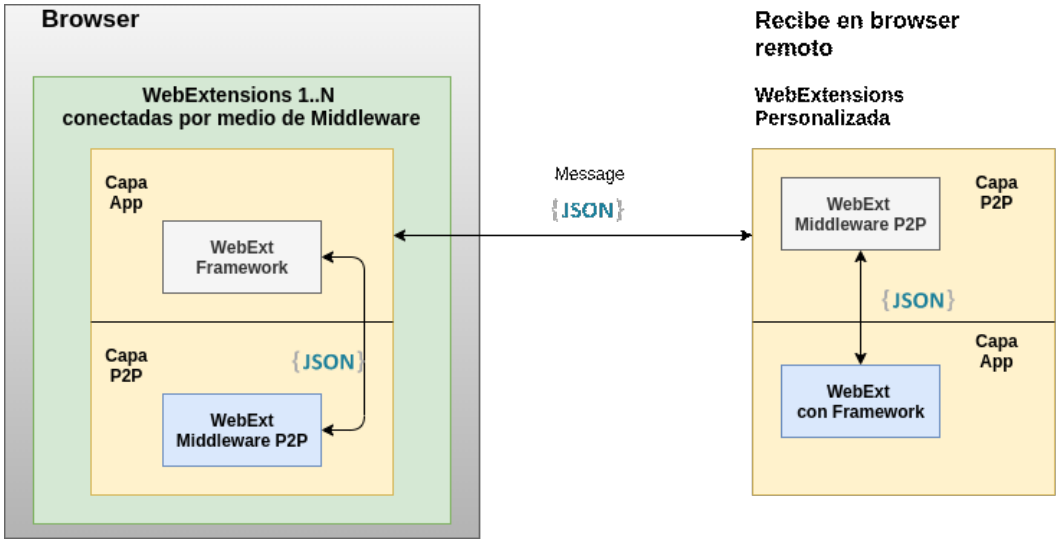


Figura 4.3: Interacción entre mensajes por medio de JSON entre Middleware y Framework.

Como se puede apreciar en la figura 4.3, tenemos dos capas. Por una lado, tenemos la capa de aplicación en sí misma que provee la WebExtension personalizada, mientras que, en la capa P2P se encarga de proporcionar los métodos necesarios para ser utilizadas por la capa de aplicación. Hay que tener presente que las capas P2P de la WebExtension personalizada se deben ejecutar en el contexto background. Desde el contexto background facilita el acceso a las funcionalidades P2P.

Por lo tanto, desde una WebExtension personalizada el usuario puede utilizar UI como: popup, browser_action y sidebar, que en general son las

más utilizadas. Además, libera al usuario para poder agregar funcionalidades necesarias sobre su WebExtension.

Capítulo 5

Middleware y Framework

5.1. Middleware

La interacción entre el Middleware y el Framework permite fusionar la capacidad de P2P en las WebExtensions personalizadas. Generando un ambiente P2P donde el desarrollador de WebExtensions no tenga que preocuparse de la capa de comunicación de entre Browsers; solo en las respuesta, recepción y procesamiento de los mensajes remotos. Como se especificó en el capítulo IV 4.1, el Middleware P2P permite gestionar las conexiones entre los Browsers.

Por lo tanto, algunas de las responsabilidades que se le delegan serán:

- Se provee una pequeña UI que permite interactuar con los mensajes del Middleware y con los mensajes de las WebExtension personalizadas. A partir de la UI, se permitirá ver los mensajes en cola; aceptarlos o rechazarlos.
- Se provee una interfaz a través de canales de comunicación que permiten enviar y recibir mensajes desde y hacía las WebExtensions.
- Se proveen dos metodologías de conexión. Una a través de internet y otra a través de una LAN. Dependiendo del tipo de conexión, cada Browser podrá enrutar los mensajes de forma independiente al tipo de red que ofrezca el sistema operativo.
- Se provee la modalidad de uso como: Hybrid, server, client, Alone y Manager. En modo client, solo envía mensajes y no procesa las recepciones; mientras que, en modo server, puede recibir y procesar los

mensajes para luego enviar las respuestas. El modo híbrido es por default y permite al usuario operar en ambos modos. En Manager y Alone es para propositos de testing y desarrollo.

Las WebExtensions pueden colaborar y reducir el tiempo de procesamiento a través de P2P. Por ejemplo, una WebExtension podría procesar un cálculo hasta la posición n mientras, otro Peer puede seguir procesando esos resultados hasta la posición n+1. Está funcionalidad podría enviar los datos, acompañado por la implementación que se utiliza para procesar o solo enviar los datos, similar a una funcionalidad de forward. La delegación de tareas es un acercamiento hacia la cooperación de recursos(cómputo y almacenamiento).

Actualmente, para que todo funcione correctamente, se requiere que estén instaladas en el mismo Browser cada WebExtension P2P (el Middleware) y la WebExtension personalizada. Ya que la comunicación se realiza a través de un conector externo que se denomina runtime.onConnectExternal. Este conector permite interactuar entre las WebExtensions, facilitando el canal de comunicación y, a su vez, permite conectar múltiples WebExtensions a través de n puertos de conexión. El Middleware contiene un listener[1] que permite recepcionar cada mensaje que es enviado desde la WebExtension personalizada y responder solamente a aquella que realizó la petición. Cada comunicación se genera a través de un identificador único que provee cada WebExtension.

Cabe destacar que los mensajes se irán encapsulando de forma similar al modo de funcionamiento de un paquete de redes[2]. De esta forma, al llegar a destino, se lee y verifica que el mensaje corresponda a una WebExtensions instalada en el Browser. Luego, se realiza un reenvío hacia la WebExtension correspondiente y, finalmente, será ella la que realice el procesamiento que corresponde.

Cada mensaje desde, y hacia las WebExtension, son encapsulados utilizando JSON. A partir del encapsulamiento, permite traspasar cada capa de procesamiento, siendo el objeto más interno aquel relevante para procesar o leer.

[1] En este caso se refiere al Evento sobre un puerto; es decir está atento a la llegada de mensajes, cuando arriba el mensaje, se ejecuta las acciones correspondientes.

[2] Hace referencia al Modelo OSI, donde los paquetes se van encapsulando desde la capa de aplicación hasta la capa física.

El Middleware, entre otras de sus funcionalidades, permite la opción de utilizar el envío de scripts entre usuarios, sin tener ninguna WebExtension personalizada instalada. Su uso, en un principio, se desarrollo a fin de realizar pruebas y, luego, se dejó como una funcionalidad nativa para aquellos usuarios que tengan solo el Middleware y quieren realizar pruebas de envío de script remotos. Cuando llega un mensaje que se encapsula en un JSON, para que lo ejecute el usuario, este debe dirigirse hasta el browseraction y seleccionar *aceptar* el mensaje; posteriormente, este se ejecutará. Más adelante, en la sección de ejemplos, se representará la funcionalidad.

5.1.1. Uso de la señalización en el Middleware

La elección de WebSocket, en la construcción del señalizador, en lugar de XHR, permite realizar la personalización de eventos, minimizando la complejidad del desarrollo y mejorando la operatividad relacionada con la conectividad del Browser. Después de todo, minimiza el impacto en la latencia de conexión entre browser y el ISP, con lo cual, permite tener un mejor desempeño entre pares [22]. Con WebSocket se genera un canal activo-activo entre el cliente-servidor, permitiendo al servidor poder generar eventos ante cambios que puedan influir en el cliente. De forma tal que, el cliente, no tendría que preocuparse explicitamente por realizar polling[3] sino que solo debería escuchar[4] los eventos de cambios para poder procesar la información actualizada.
A continuación, los ítems que se tienen en cuenta:

- Es un protocolo independiente, basado en TCP y utiliza WSS. El handshake se realiza a través de HTTP.
- Penalización mínima.
- Soportado por los browsers y aplicaciones externas por medio de librerías.
- Es asíncrono, full-duplex, orientado a eventos y mantiene el canal de comunicación abierto hasta que alguien lo cierre. Esto permite el envío de múltiples mensajes, sin tener que generar una nueva conexión.

[3] Está modalidad evita las multi peticiones (polling) del protocolo HTTP entre cliente-servidor

[4] en referencia al evento Listener de evento.

Uno de los hechos relevantes del uso de WebSocket en el señalizador fue tener control de las conexiones/desconexiones de los Browsers. Permitió, que al conectarse un Browser, envíe información al resto de compañeros conectados. De esta manera, se genera un broadcast donde cada Browser obtiene información del resto. Cuando un Browser se desconecta por algún evento, permite al señalizador tener información y enviarla al resto de los Browser. Está modalidad de conexión es necesaria ya que no se cuenta con una red paralela antes de la P2P, establecida entre los Browsers y, mientras ella no exista, es necesario tener control de las conexiones.

El uso de WebSockets mejora la respuesta del usuario y permite la personalización de eventos. Por razones de simplicidad, esta última, esta entre las más atractivas funcionalidades.

5.1.2. Middleware y WebRTC

La integración entre los contextos y scripts del Middleware y WebRTC del Browser son el núcleo de la comunicación y funcionalidad P2P.

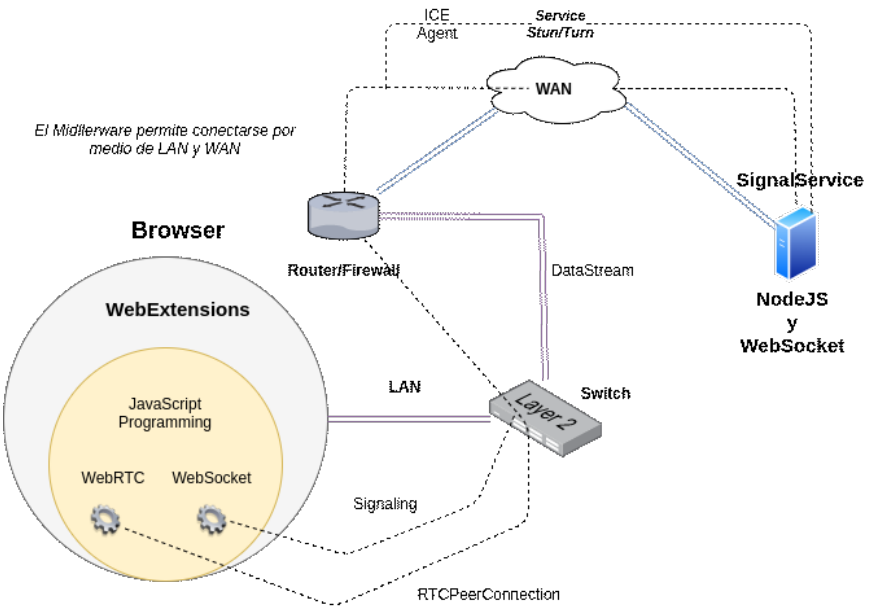


Figura 5.1: Middleware via WebRTC.

En la figura 5.1 se puede observar la integración entre la WebExtensions

como parte del Browser y las tecnologías necesarias, también el servicio del señalizador que se comunica mediante Websocket. Por lo tanto, se puede observar la existencia de más capas que interactúan hasta llegar al usuario remoto, mientras que el usuario solo usa la capa más abstracta de la comunicación. El Middleware usa la API del Browser que permite interactuar con WebRTC y así poder delegar la funcionalidad necesario para poder conectarse. Si el browser se actualiza, en un mejor caso, al ser un estandard, no habrá que realizar cambios y, en un peor caso, el cambio deberia ser de menor impacto.

5.1.3. Middleware como WebExtension

En cuanto al funcionamiento de la arquitectura, hay que tener en cuenta que dispone de varias funcionalidades a la hora de generar la comunicación entre los contextos background, content-script y cada UI (browserAction y options).

A continuación se detallan algunos de los tipos de archivos de scripts que son utilizados en la aplicación.
Solo se tendrá en cuenta aquellos que tienen relación con la arquitectura de desarrollo utilizada para la WebExtensions del proyecto.

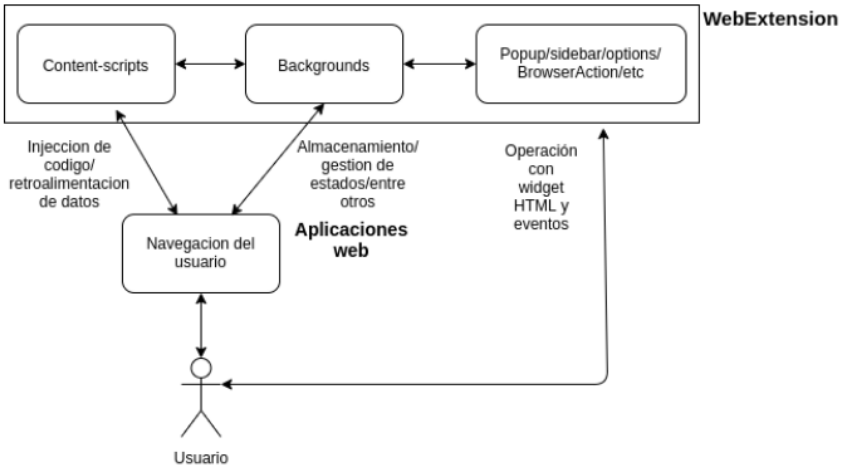


Figura 5.2: Componentes que se utilizan en la WebExtension.

Background: es parte del contexto de persistencia. En este contexto, es posible utilizar funcionalidades que acompañan durante toda la vida de la WebExtension. Es decir, el usuario puede navegar de forma transparente mientras los datos, que utiliza la aplicación, seguirán vigentes aún si los tabs son cerrados. Permite preservar los estados frente a cambios en la página web o tabs. Este deja de funcionar al cerrar el browser o al deshabilitar/desinstalar la extensión y se inicia al instante de ser cargado, por medio de la acción del usuario.

Popup page: es una funcionalidad que permite tener una página web que se despliega por medio de acción del usuario, al interactuar con un botón propio de la extensión. A partir de ella, es posible interactuar con contenido del content-script y background. Por ejemplo, accionar eventos del browser que permitan interacción con la página que se encuentra navegando el usuario.

Options page: permite agregar una página web con opciones de configuración de la webExtension. Está funcionalidad permite programar una página a medida, a fin de parametrizar los parámetros de la aplicación.

Content scripts: está funcionalidad permite ejecutar scripts programados durante la carga de páginas que visita el usuario con el browser. A su vez, permite modificar el comportamiento de la página, permitiendo al usuario personalizar el comportamiento de la web según sus necesidades. Es decir, permite interactuar con los elementos del DOM.

BrowserAction: es un botón que se encuentra en la barra de herramientas del browser. Permite agregar el evento click, a fin de visualizar el popup page.

ContextMenus: está funcionalidad permite generar un menú de opciones para el usuario del browser. Permitiendo emitir una cantidad de finita de opciones programables, con cada click que el usuario interaccione.

Sidebar action: es un panel que se despliega del lado izquierdo del browser. También es una página web que permite obtener estado y variables desde el background, así como accionar eventos por medio de APIs. En el proyecto es utilizado con el propósito de gestionar una herramienta para Testing.

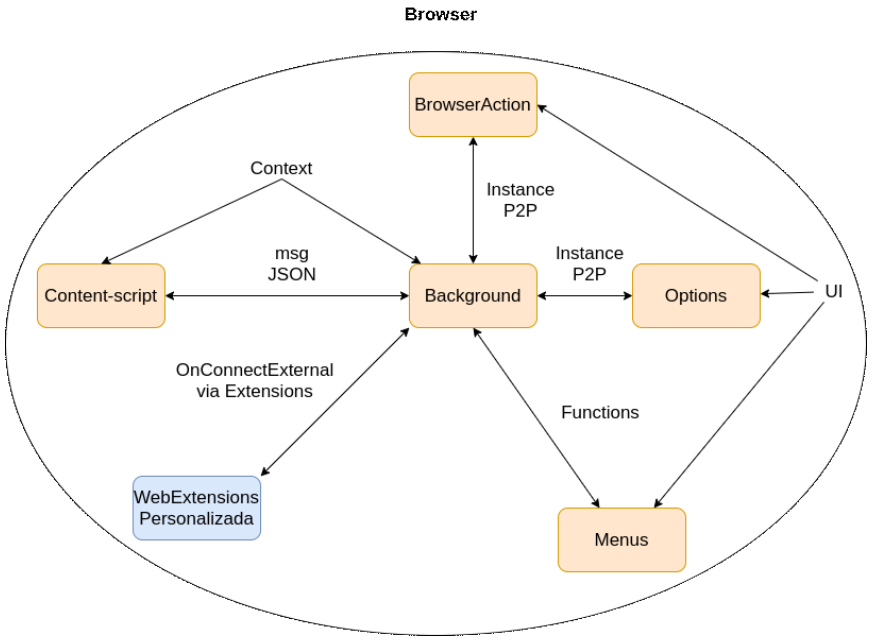


Figura 5.3: Browsers y relacion con WebExtension.

En la figura 5.2 se representan algunos de los compontes que se han descripto 5.1.3 y la relación con el usuario. Mientras que, en la figura 5.3, se pueden observar los contextos y componentes involucrados con sus relaciones de uso en la webExtension del proyecto dentro del browser. Cada uno de ellos es un archivo en JavaScript, el cual es configurado por medio de un archivo de configuración que se denomina *manifest.json*. Este archivo es el encargado de unir todos los archivos en un solo lugar y, a partir de allí, el Browser va a generar la carga respectiva de la aplicación, cargando las configuraciones y cada dependencia de cada archivo particular.

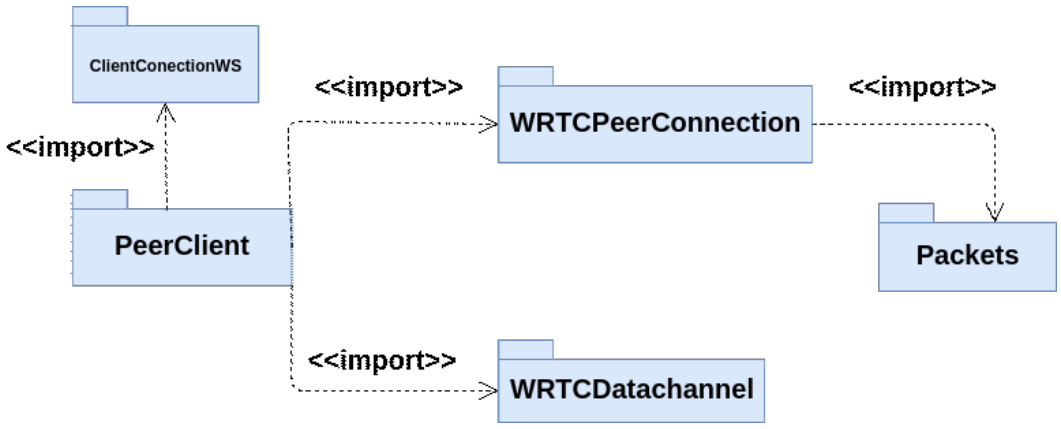


Figura 5.4: Diagrama de paquetes del Middleware.

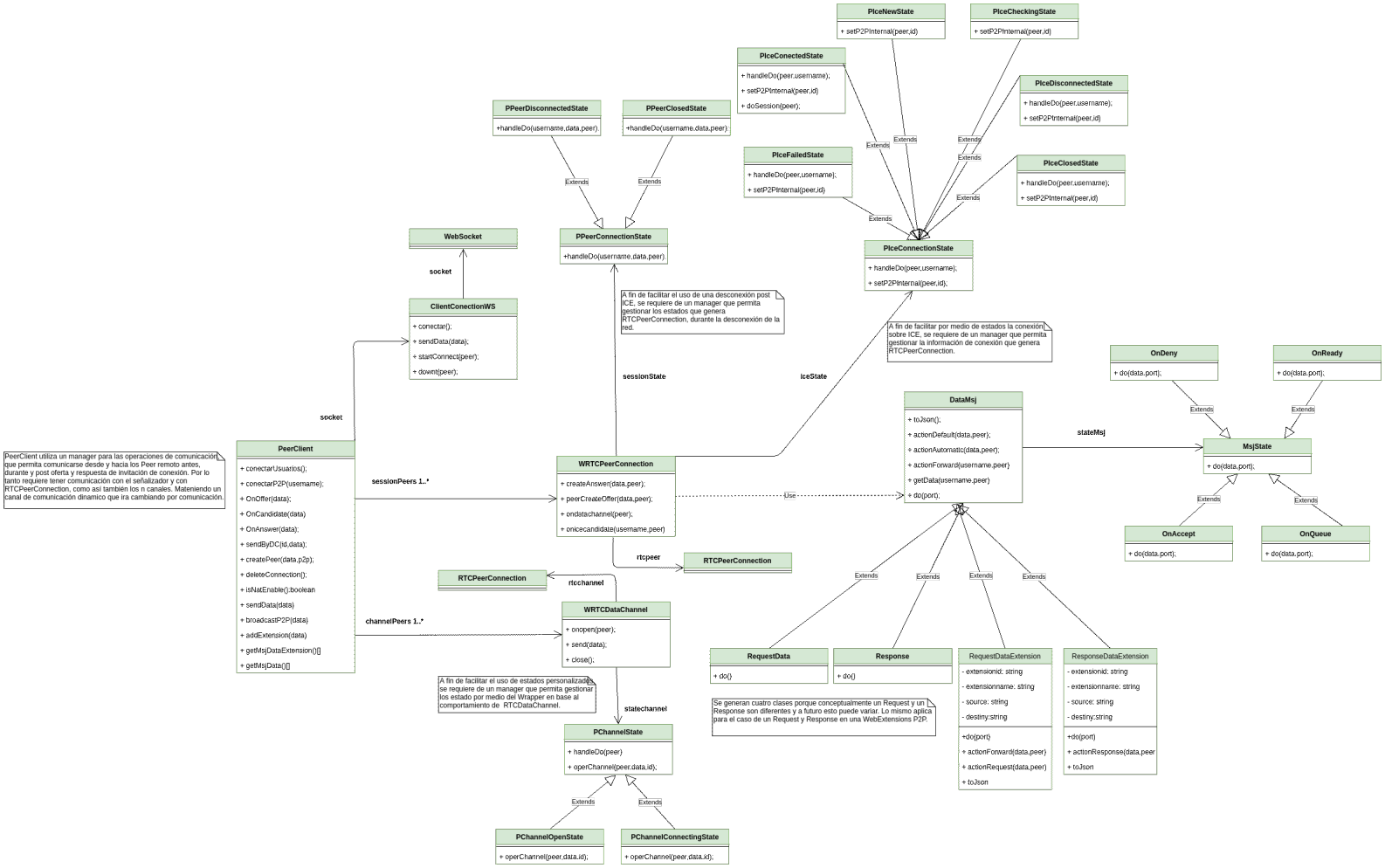


Figura 5.5: Diagrama clases del Middleware.

5.2. Framework

La funcionalidad del Framework permite al usuario abstraerse de la complejidad de utilizar la comunicación P2P. Su lógica de comunicación con el Middleware, para el envío y recepción de mensajes, se realiza en el contexto background de las WebExtensions. Desde el contexto background, facilita el acceso a las funcionalidades P2P del Middleware. Por ejemplo, desde una UI popup, browser_action o sidebar, se puede acceder al objeto que referencia a las funcionalidades del Middleware y realizar llamadas a funciones por medio de callback.

La recepción de mensajes se realiza por medio de un listener que mantiene la conexión con la extensión del Middleware; en ella, se programan los tipos de mensajes y las acciones que realiza según cada caso. La WebExtensions personalizada debe de heredar de AbstractP2PExtension (nombre de la clase abstracta) e implementar las estrategias, por medio de algoritmos, para el envío y la recepción de mensajes.

La capacidad de delegar la funcionalidad de comunicación y centrarse en la lógica de lo que proporciona la WebExtension personalizada, permite al usuario que solo tenga que preocuparse por programar lo necesario para la llegada de un mensaje hacia su WebExtensions. En esta primera aproximación de implementación del Framework son, al menos, tres funciones básicas que otorgan valor agregado, en relación a la funcionalidad P2P.

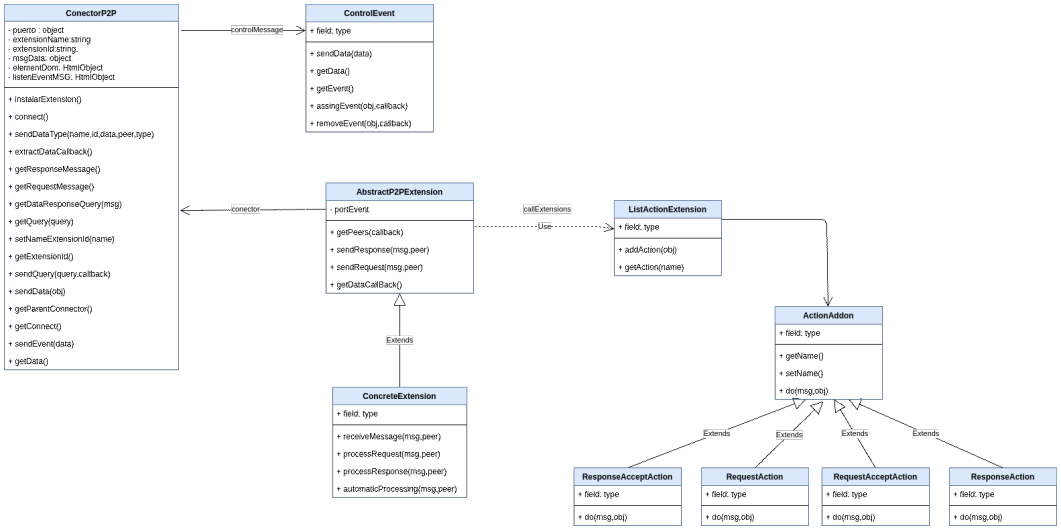


Figura 5.6: Diagrama de clases.

Como se puede observar en la figura 5.6, la clase ConcreteExtension tiene los métodos:

- **processRequest**: es el encargado de realizar la acción cuando se recibe el mensaje. La acción se relaciona con un mensaje de recepción de request.
- **processResponse**: es el encargado de realizar la acción cuando se recibe un *response*. Es decir, mostrar un mensaje de recepción de response.
- **receiveResponse**: es la respuesta que retorna del procesamiento realizado por el peer remoto.
- **automaticProcessing**: es un agregado que permite responder de forma automática, sin necesidad de requerir asistencia por parte del usuario; es decir, sin pasar por processRequest.
- **Connect**: no debe ser implementado en la nueva WebExtensions, sólo debe utilizarlo desde el background en su primera línea, luego de generar la instancia de la clase concreta.

Hay que tener en cuenta que las implementaciones deben de ejecutarse en background y, en caso de utilizar contenido fuera de él, tomando co-

mo ejemplo content-script, se requerirá realizar un canal de comunicación entre el content-script y background. Mientras que, si utilizamos browser action, options y page actions popup, sería posible acceder a la instancia a partir de las funciones disponibles en una WebExtensions como: *browser.extension.getBackgroundPage()*.

```
let backgroundInstance = browser.runtime.getBackgroundPage();
let P2PExt = backgroundInstance.P2PExt;
```

En el código se observa el acceso a la instancia P2Pext (nombre que representa una instancia que implementa el comportamiento de *AbstractP2PExtension*). Desde la variable P2PExt, es posible acceder a las funcionalidades y variables que se encuentran dentro del contexto background. Si se requiere acceder a la funcionalidad desde un content-script, es necesario generar un medio de comunicación, ya que no es posible acceder al scope del background desde el content-script.

Queda a disposición del usuario el alcanse de la funcionadlidad de la WebExtension porque un Browser permite utilizar las WebExtensions convencionales y construir WebExtensions por medio de paquetes, agregando complementos de tecnologías a partir de herramientas como WebPack (no está dentro del alcance del documento explicar dicho uso de paquetes). Por lo tanto, utilizar WebPack, en su rol de empaquetador de tecnologías permitiría al usuario agregar nuevos paquetes que enriquezcan la UI del usuario y, al mismo tiempo, agregar ventajas de funcionalidades, a partir de librerías de terceros.

El usuario debe tener en cuenta que, para poder interactuar y funcionar en modo P2P, será necesario que puedan establecer un canal de comunicación entre la instancia del objeto, que se encuentra el en background, y aquellos objetos que residan en la UI que fueron generados por medio de WebPack. Por ejemplo, al utilizar un framework de frontend, este se puede ejecutar en el contexto de content-script y deberá generar un canal de comunicación entre este y el background. Al utilizar un evento click en un botón, este puede guardar el dato en una variable que, luego, es enviado al background por medio de un mensaje. Cuando el mensaje llega al background, se procesa y lo envía al peer destino.

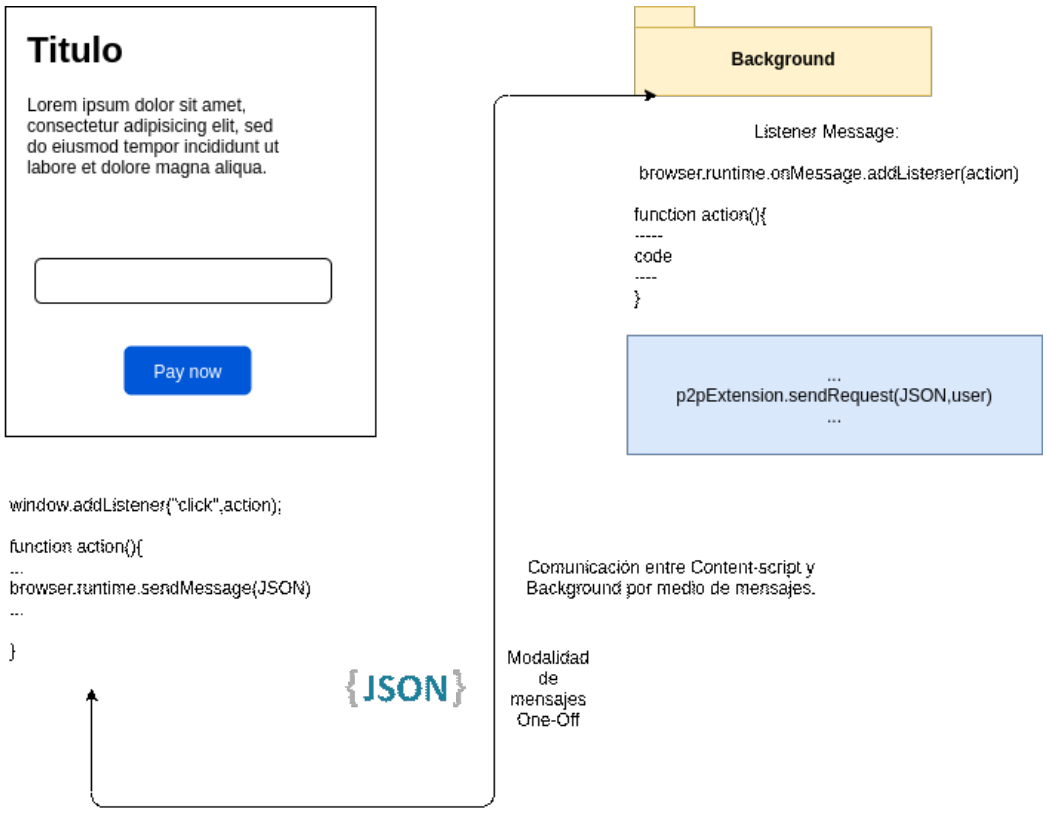


Figura 5.7: WebExtension y la comunicación entre Framework y Middleware.

En la figura 5.7 se puede observar un ejemplo de comunicación sin utilizar el método *browser.extension.getBackgroundPage()*. El Framework de frontend envía un mensaje a partir del evento de click; la información viaja por un canal de comunicación que se genera entre Content-Script y Background. Cuando se recepciona desde un listener, este envía su contenido utilizando la instancia P2PExt. En el ejemplo se puede apreciar una modalidad de envío de mensajes de patrón *one-off messages*[5]; sin embargo, también es posible utilizar una modalidad de envío de mensajes denominada Connection-based messaging[6], utilizado para soportar una mayor cantidad de mensajes.

[5]En este patrón de mensajes, se utiliza una cantidad minima de mensajes, al menos se espera una respuesta y un tiempo de vida corto. No hay elevada interacción.one-off

[6]Permite operar a traves de un puerto o varios, de forma tal que es posible maximizar la cantidad de mensajes, mejorando la calidad envio-recepción.

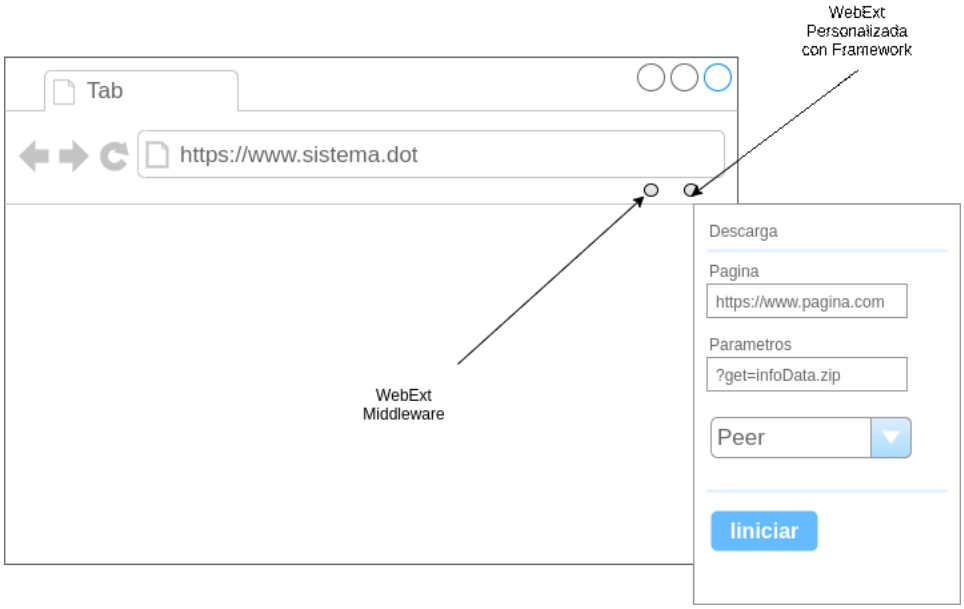


Figura 5.8: GUI de BrowserAction en WebExtension personalizada.

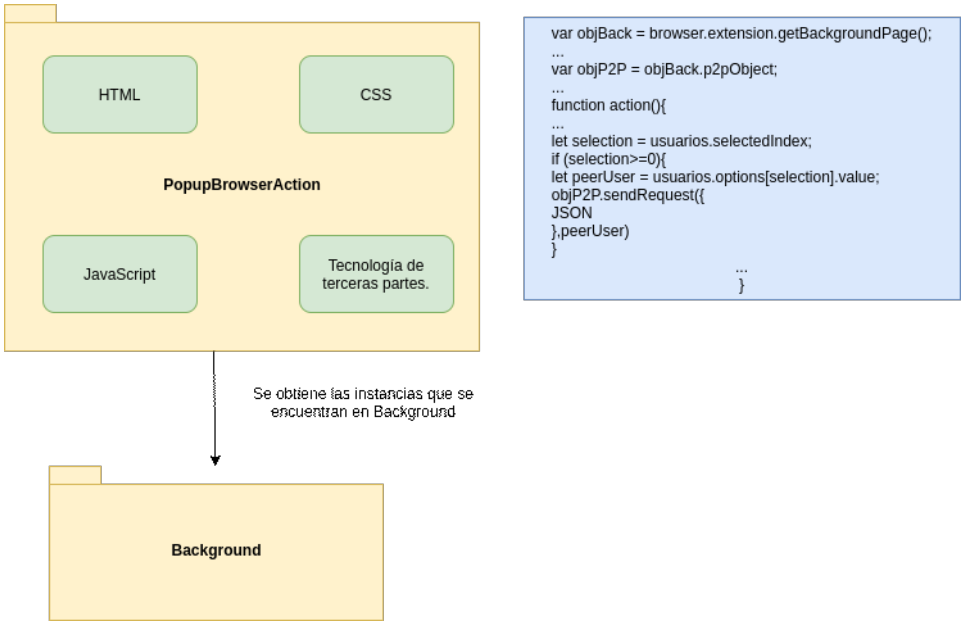


Figura 5.9: BrowserAction y obtención de la instancia del objeto que usa el Framework.

En la figura 5.9 se puede observar la funcionalidad al utilizar una WebExtensions personalizada, a partir del uso de un browserAction. A diferencia del anterior[7], no es necesario generar un canal de comunicación entre la WebExtensions desde el Content-Script hacia Background porque es posible acceder al scope del Background, por medio de una función de WebExtensions 5.2.

```
"background": {
    "scripts": ["FP2P.js","CustomExtension.js"]
}
```

El archivo FP2P.js representa el Framework que permite comunicarse con el Middleware, a partir de un canal de comunicación. Mientras que CustomExtension.js representa la funcionalidad de la WebExtensions personalizada, a través de la clase ConcreateExtension, que utiliza la función P2P. En resumen, la WebExtensions personalizada requerirá implementar en el

[7]relacionado con el acceso content-script y background.

archivo CustomExtension.js todas las funciones que crea conveniente para su finalidad.

```
Class CustomP2P extends AbstractP2PExtension {

     processRequest(msg,peer)...
     ...
     processResponse(msg,peer)...
     ...
     automaticProcessing(msg,peer)...
     ...
     receiveResponse(msg,peer)...
}
```

El código anterior, implementa la sección de código del cascarón de la implementación dentro del archivo CustomExtension.js.Como se puede observar, el usuario solo debe implementar las funciones acordes a las responsabilidades de su WebExtension.

Capítulo 6

Construcción del ambiente de prueba y casos de uso

6.1. Casos de uso HelloWorld con el Middleware

En está sección se explora el ejemplo HelloWord con el Middleware. Es este escenario, los browsers generan una comunicación P2P, por medio de una red LAN. Una vez que la comunicación se ha establecido, se procede a realizar un intercambio de codificación en lenguaje JavaScript. El objetivo es mostrar la funcionalidad nativa que provee la WebExtension instalada en el Browser.

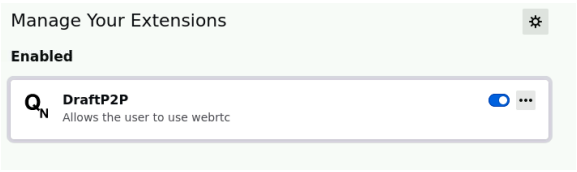


Figura 6.1: Primera pantalla desde el panel de addons disponibles en el Browser.

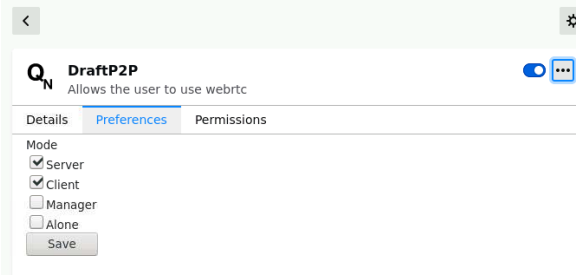


Figura 6.2: Opciones disponibles del Middleware como WebExtension.

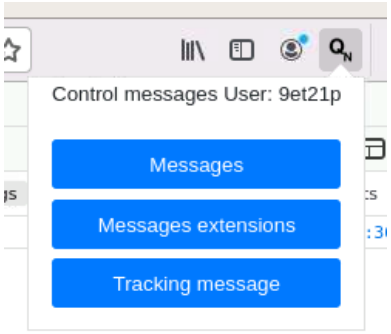


Figura 6.3: Usuario aleatorio 9et21p con el Middleware instalado en el browser.

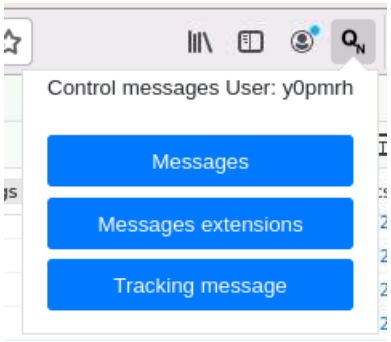


Figura 6.4: Segundo usuario aleatorio y0pmrh.

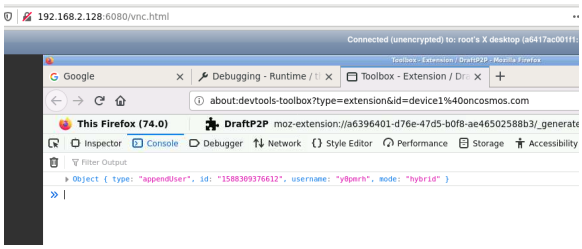


Figura 6.5: Reporte de consola en browser con ip final 128.

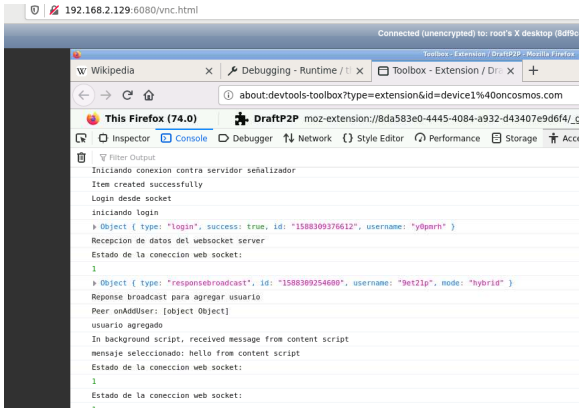


Figura 6.6: Reporte de consola en browser con ip final 129.

Como se puede observar en las figuras 6.1 y 6.2, por default, se encuentra instalado en modo híbrido (cliente-servidor). Además, cuenta con dos opciones más como *Manager* y *Alone*, que permiten una funcionalidad diferente. Por un lado, existe el modo manager que se utilizó para propósitos de testing; como por ejemplo, para coordinar peers y poder ejecutar comandos remotos. Mientras que, la opción Alone, es agregada con el propósito de desarrollo de WebExtension en modo P2P. A partir de ella, un programador pude desarrollar toda la funcinalidad P2P con JavaScript, HTML5 y CSS, sin necesidad de estar conectado a la red. También posible interactuar con todo el desarrollo de una WebExtension; simulando los request y response de cada operacion que se necesite, permitiendo utilizar la completitud del Framework conectado al Middleware. También se puede apreciar que se generan dos usuarios, cada uno es contruido con un nombre de string aleatorio 6.3, que se realiza de forma automática (por simplicidad al funcionamiento)

en cada browser y mantiene una relación con el servicio de señalización. En las figuras 6.5 y 6.6, se muestra información de la operación que se realiza en el browser con la interacción del servicio de señalizador.

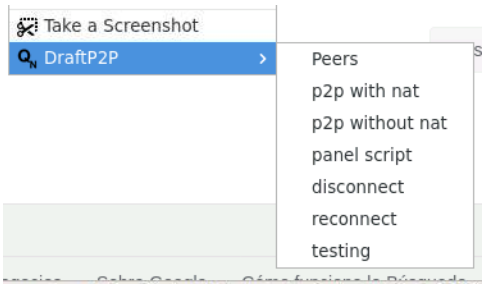


Figura 6.7: Opciones disponibles de la WebExtension.

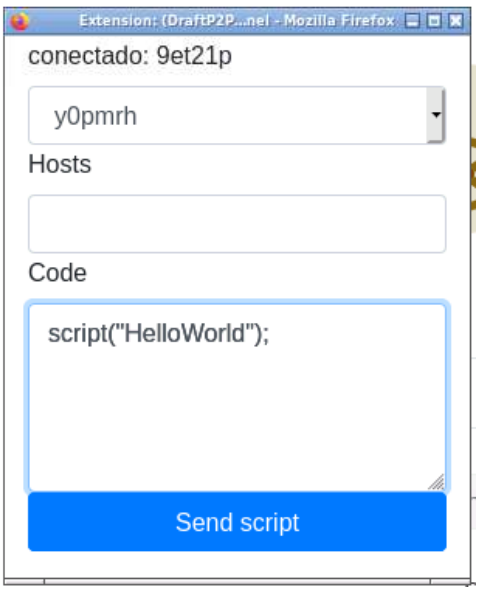


Figura 6.8: Pantalla de codificación de script.

Para el escenario de prueba de HelloWorld vamos a realizar la conexión *P2P without nat* (para nuestro caso de uso de prueba con NAT, y sin ella, es lo mismo). Cuando se realiza la conexión directamente, se procede a realizar un ofrecimiento de conexión entre los browsers y, desde allí, ya interviene

WebRTC con el servicio de señalización. Como se puede observar en el menú contextual 6.7, hay una opción definida como **Panel script**. Al realizar click en ella, se despliega una ventana como la presentada en la figura 6.8.

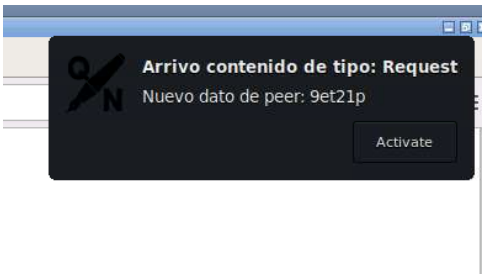


Figura 6.9: Mensaje de arribo.

Dentro del panel de codificación, se encuentra un script de ejemplo que realiza un *alert*. Está función será enviada hacia el peer remoto y, luego, cuando sea recepcionada y aceptada, se procederá a ejecutar en el browser del cliente remoto. Se puede apreciar que en cada mensaje que llega desde un browser remoto hacia otro, se despliega un mini popup que avisa que arribó contenido e informa desde que peer 6.9

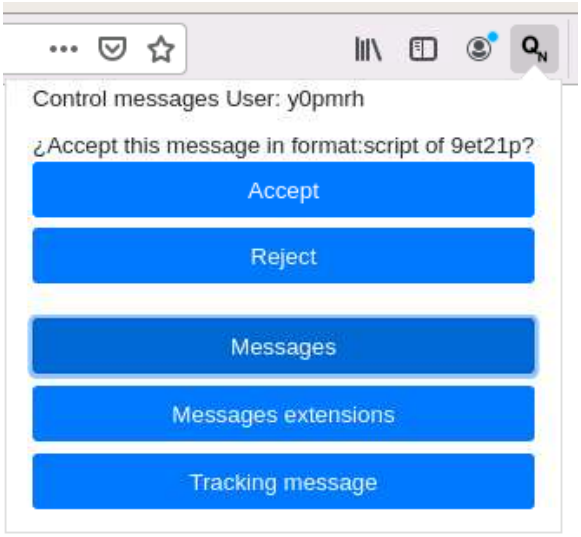


Figura 6.10: BrowserAction con las opciones disponibles.

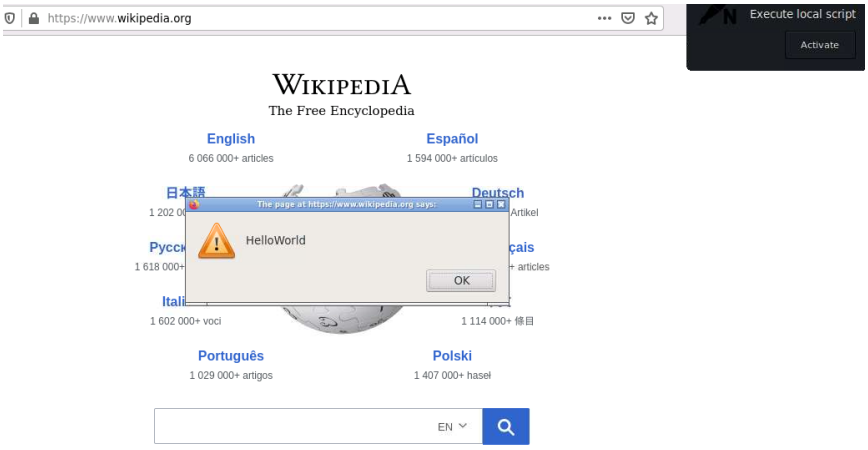


Figura 6.11: Alert ejecutado en el broser del peer remoto.

Una vez que llega un mensaje, es el usuario remoto el que tiene que proceder para aceptar o denegar el mensaje. Como se puede observar en la figura 6.10, al realizar click en aceptar, se procede a ejecutar código JavaScript en el browser cliente. La ejecución registra el código en el browser remoto y, luego, realiza la acción con ayuda del Content-Script 6.11.

La función nativa del Middleware permite una modalidad como el envío y recepción de scripts remotos. Interactuando con la página del cliente, así como informar qué está observando el usuario y, luego, enviar dicho procesamiento a un servicio en la nube o, simplemente, almacenarlo. Es una funcionalidad simplificada de lo que se puede obtener con los scripts que nos proveen los repositorios como greasyfork y userscripts-mirror. Con la diferencia que los usuarios, son los que la generan en tiempo real y los envían a otros Peers que deseen utilizarlo. Por ejemplo, dos programadores pueden tener un código de frontend que están probando y en lugar de enviarse el código por medio del chat, tienen la posibilidad de realizar la prueba en tiempo real.

6.2. Casos de uso Middleware y Framework

En este escenario se presenta la funcionalidad WebExtensionP2P que usa el Framework como medio para el envío de mensajes. En este escenario se eligió un ejemplo, donde, en ambos browsers, tienen instalada una WebEx-

tension al que denominamos *SearchP2P*. Una vez que ambos se encuentran instalados y conectados P2P, se procede a solicitar una búsqueda en uno de los motores disponibles. Cuando el usuario remoto recibe una petición de búsqueda, este la acepta, lee un string y realiza la búsqueda en el motor que se solicitó.

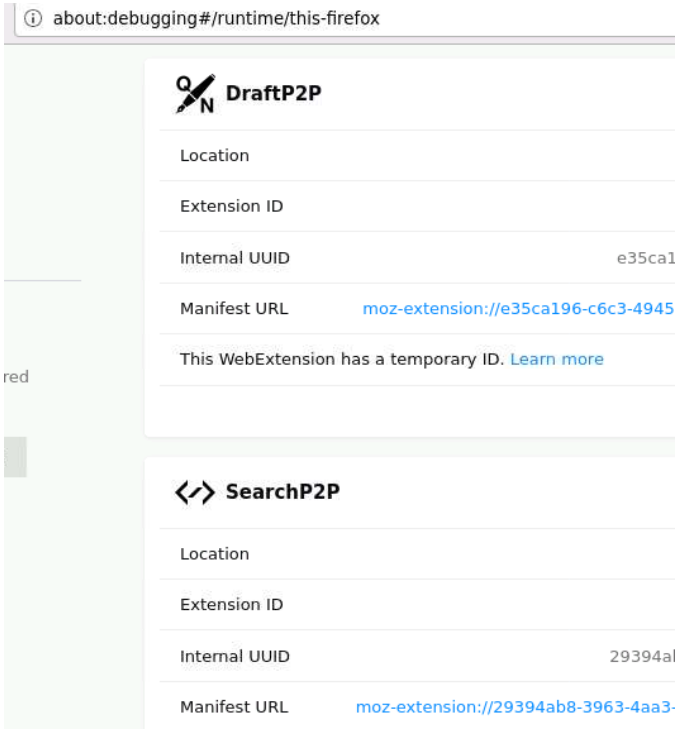


Figura 6.12: WebExtensionP2P en conjunto con el Middleware.

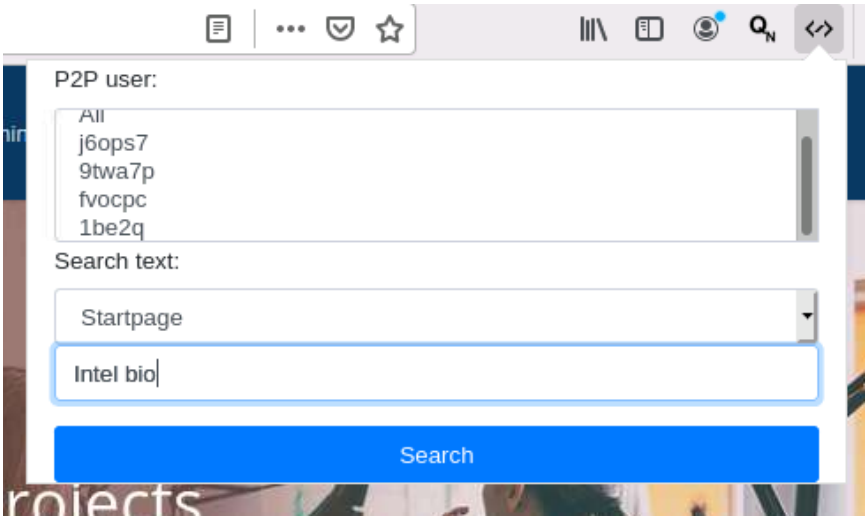


Figura 6.13: SearchP2P desde BrowserAction.

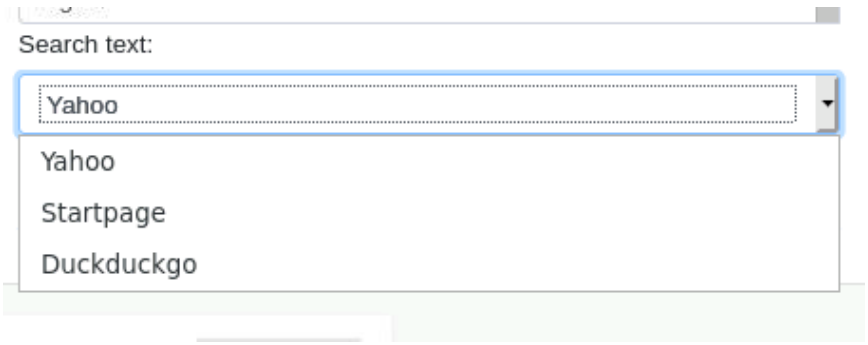


Figura 6.14: Opciones de motores de busqueda disponibles.

Como se puede apreciar en la figura 6.12, ambas WebExtensions estan instaladas. Por un lado el Middleware y por otro la WebExtension personalizada que usa el Framework (SearchP2P), y en la figura 6.13 se puede observar más de un peer conectado. Por lo tanto, es posible enviar el mensaje a todos o solo a uno de ellos. En el inputbox 6.13, el usuario puede introducir el texto que necesita buscar. Y en la figura 6.14, el usuario selecciona el motor que quiere que utilice el peer remoto para realizar la busqueda.

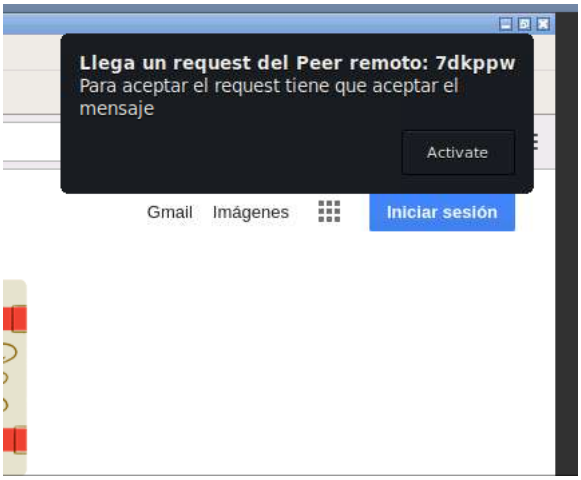


Figura 6.15: Aviso del mensaje de recepción.

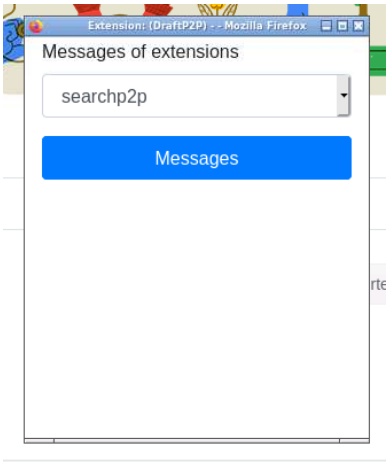


Figura 6.16: SearchP2P desde el panel de mensajes.

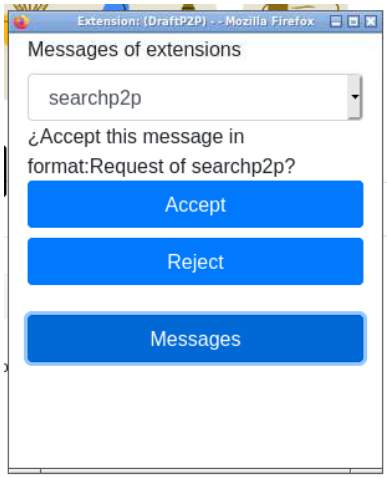


Figura 6.17: SearchP2P popup para aceptar o rechazar el mensaje.

Cuando recibe el mensaje por parte del peer remoto 6.15, es el peer destino quien debe proceder a realizar la acción de aceptar el mensaje. Para ello, necesita utilizar el browseraction desde las opciones disponibles del Middleware. Una vez que realiza click, debe seleccionar la opción *Message extensions* y se desprende un popup como la figura 6.16. A diferencia de la opción mensajes del Middleware, en este caso solo vamos a ver los mensajes que se relacionan con las WebExtensionP2P (searchP2P) que se encuentran instaladas y utilizando el Framework. Cuando el usuario acepta el mensaje 6.17, se ejecutara la acción predeterminada relacionada con SearchP2P.

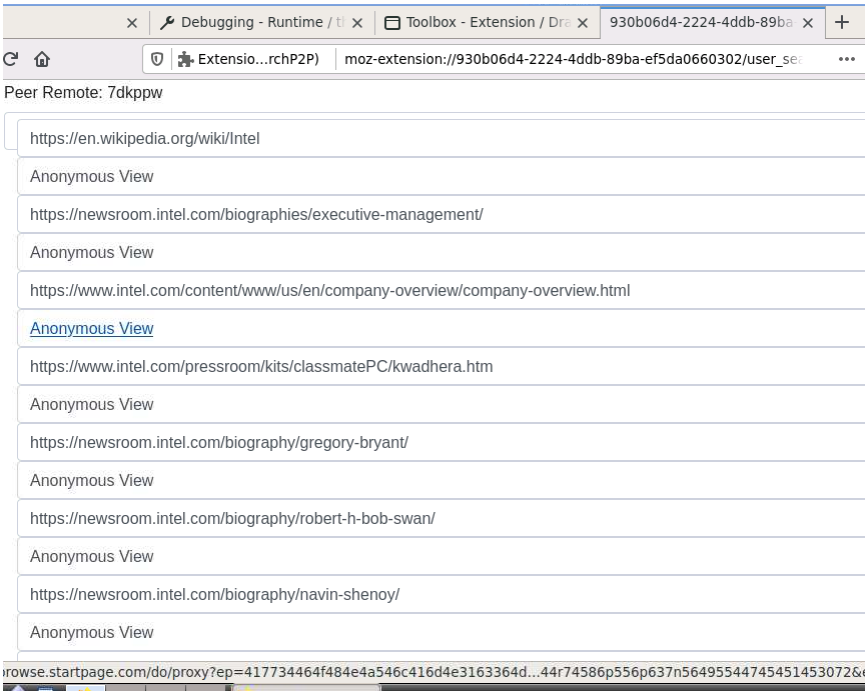


Figura 6.18: Resultado de la busqueda.

Como se puede visualizar, el resultado de la figura 6.18 corresponde con la búsqueda del string solicitado, en el motor de búsqueda seleccionado, en este caso Startpage. Ahora, el peer remoto se encuentra en condición de enviarlo al peer que lo solicitó. De está forma, se pueden realizar n busquedas en diferentes motores, a fin obtener y generar un Mashup de contenido representado con los resultados de búsquedas que cada proveedor ofrece en cada peer.

Caso de uso potencial P2P

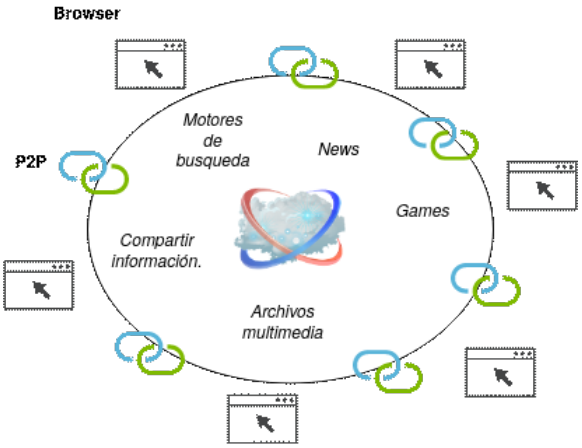


Figura 6.19: Caso de uso potencial para compartir información y funcionalidades entre peers.

Como se puede observar en la figura 6.19, este podría ser un escenario potencial de uso en relación a las WebExtensions P2P, orientadas a la búsqueda de contenido. Uno de los aspectos importantes a tener en cuenta en los contextos de búsquedas, es que la acción de buscar, al ejecutarla en repetidas ocasiones, revela información relacionada a la sesión de cada usuario (geografía, gustos, ip, etc) y sus intereses, los cuales se acumunulan en el tiempo. Por lo tanto, los resultados podrían estar alterados por la afinidad de información del mismo usuario que realiza la consulta. Entonces, al tener la posibilidad de diversificar las búsquedas en otros peers, se tiene en cuenta la posibilidad de obtener resultados libres de la influencia propia del usuario. Por ejemplo, un caso de uso de WebExtension a potenciar podría ser CYCLOSA[1], al agregar la funcionalidad P2P, se podría mejorar la descentralización de busqueda y los resultados.

[1]Es una WebExtension que permite realizar busquedas de forma descentralizada y generar una capa de privacidad [23].

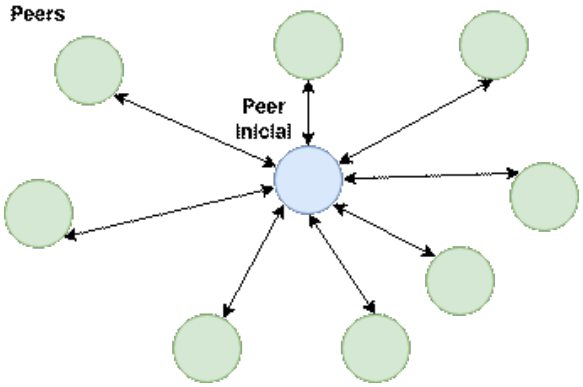


Figura 6.20: Modalidad de conexión de un peer conectado por default con la funcionalidad LAN/WAN.

En la figura 6.20, se refleja la modalidad de conexión que se genera al realizar P2P desde un cliente por default. En esta primera versión del Middleware, cada usuario que se conecte y realice la acción P2P, se conectará con todos aquellos que estén online, al menos dentro del sistema de señalización. Entonces, un usuario en lugar de realizar n actividades individuales (como búsquedas), tendría la posibilidad de realizar actividades en forma colaborativa. Por ejemplo, la descarga de archivos, extraer información, enviar información, procesar de información distribuida, realizar scrapping distribuido, entre otros.

Otro de los casos alternativos es realizar la funcionalidad de prueba de carga distribuida sobre un motor de búsqueda. Es un escenario donde cada peer de carga genera una petición desde una ubicación geográfica diferente y solicita un tópico. En este escenario, se podría medir los tiempos de respuesta y la escalabilidad de la aplicación con la demanda de peers, sin necesidad de generar grandes despliegues de infraestructura. Permitiendo probar módulos de aplicaciones que tengan APIs de acceso público o privado (con acceso previo.). Por lo tanto, así como una prueba de carga por internet también sería posible realizar pruebas a través de LAN por medio de contenedores, a través de un peer manager que gestione un grupo peers conectados, permitiendo tener a disposición un enjambre de conexiones y recursos.

6.3. Casos de uso Middleware y Framework parte II

En este caso, se experimenta con un conjunto de tecnologías empaquetadas por medio de WebPack. La WebExtensionP2P utiliza un poco más de complejidad, ya que se encuentra desarrollada por medio de librerías como JQuery, browser-polifyll, compromise.es6.js, entre otras (las cuales no son del alcance de este documento). Con dichas tecnologías, se desarrolló una WebExtension, que se denominó NewsP2P, para experimentar con la distribución de información selectiva de fragmentos de textos de la web y conocer la relevancia de palabras que en ellas se encuentran. Dicha problemática se resuelve reportando la frecuencia y porcentajes de palabras relevantes, aplicando algoritmos de *NLP (Natural Language Processing)*. A través del algoritmo se agiliza el procesamiento de palabras de un fragmento de texto. De forma tal que de una página de n párrafos, el usuario puede seleccionar todas aquellos que necesite; luego, se almacenan en un vector que será enviado para procesar al peer remoto. Cuando llegan los datos al peer remoto, esté aplica el algoritmo de NLP, que se encarga de seleccionar un conjunto de palabras relevantes con ayuda de la libreria *compromise.es6.js*, analizando cada una de ellas y también genera un vector resultante para ser enviado, con los detalles del fragmento de texto original y que palabras relevantes en ella se encuentran. En el caso de la selección de los fragmentos de textos, desde el peer origen interviene la interacción entre el background y content-script, mientras que el peer remoto solo interviene el contexto de background.

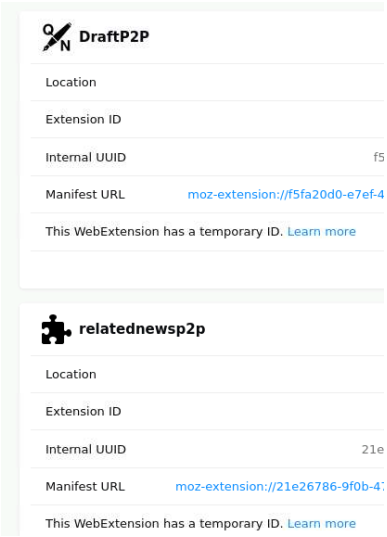


Figura 6.21: newP2P instalada en el Browser.

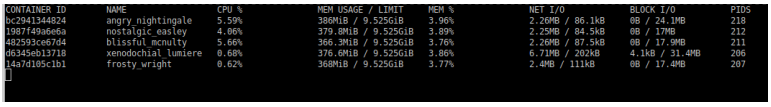


Figura 6.22: Instancias de Docker iniciadas.

Figura 6.23: Cada peer debe seleccionar el boton **enable capture**, a fin de habilitar la selección de texto.

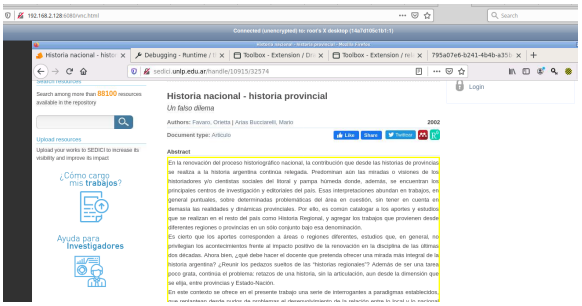


Figura 6.24: Peer uno selecciona sus fragmentos de textos.

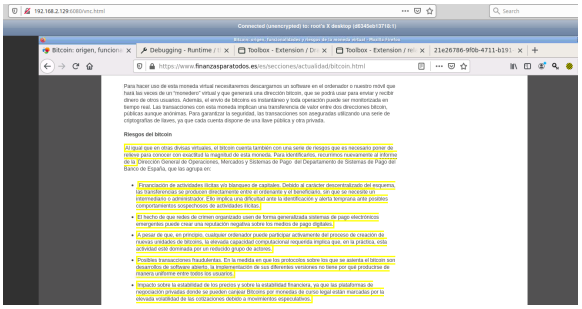


Figura 6.25: Peer dos selecciona sus fragmentos de textos.

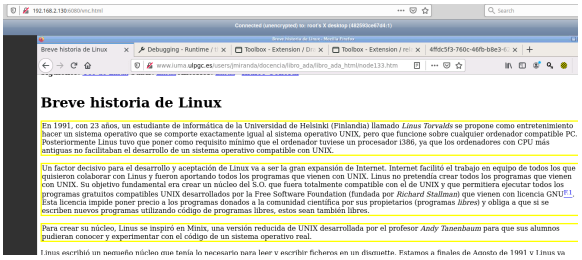


Figura 6.26: Peer tres selecciona sus fragmentos de textos.

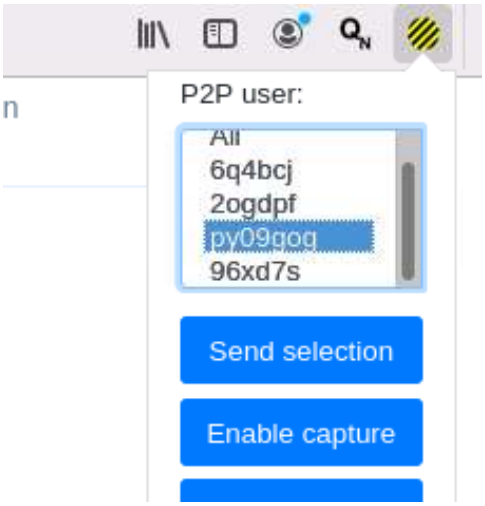


Figura 6.27: Envio de la selección hasta el peer Remoto.

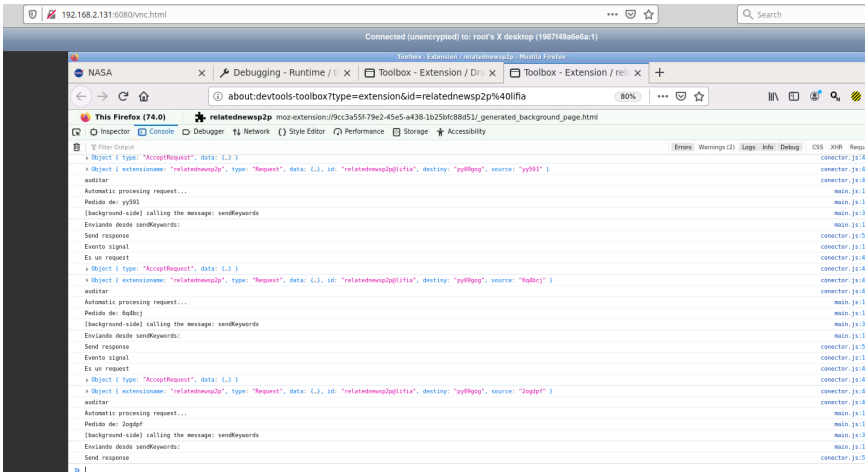


Figura 6.28: El peer que procesa las peticiones una vez que termina con cada una de ellas las envia a cada peer Remoto.

Figura 6.29: Este mensaje de recepción se presenta en todos los peers que reciben los resultados.

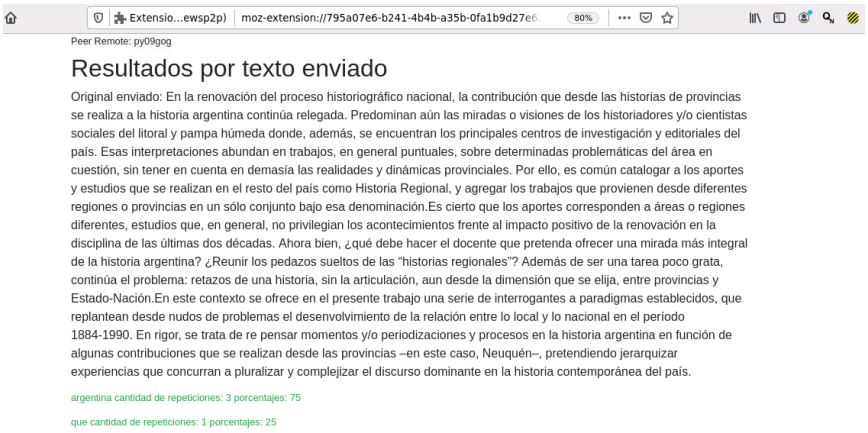


Figura 6.30: Resultados de los fragmentos de texto del Peer uno.

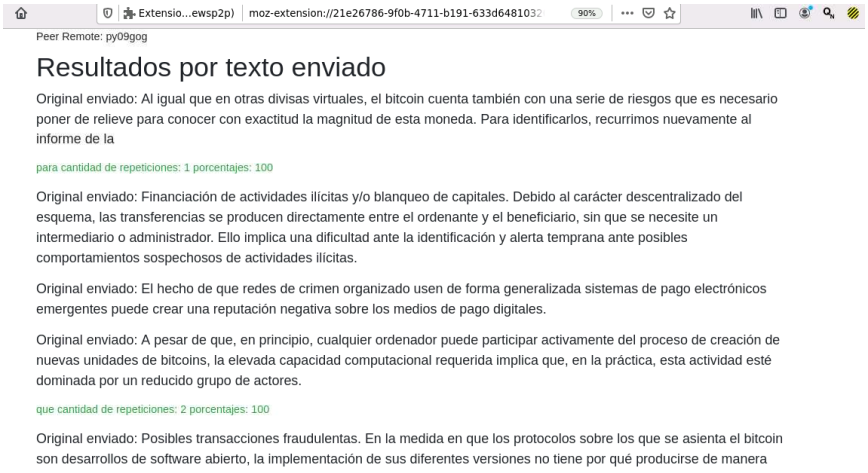


Figura 6.31: Resultados de los fragmentos de texto del Peer dos.

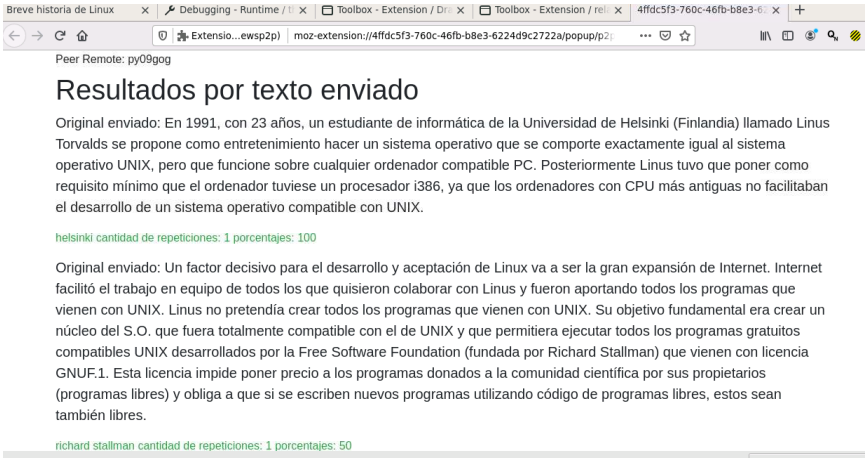


Figura 6.32: Resultados de los fragmentos de texto del Peer tres.

Para este escenario se generaron cuatro containers 6.22, de los cuales, hay tres clientes y un servidor que recibirá las peticiones de cada uno de ellos. Una vez que todos los peers instalarón la WebExtension6.21 y seleccionaron sus noticias 6.24, 6.25, 6.26, utilizando el botón del browseraction6.23, se procede a realizar en envío hacia el peer que queremos que resuelva los fragmentos de textos. En este caso, se elige el Peer **Py09gog**6.27, el cual los

procesará de forma automática porque, en este ejemplo, los *request* viajan con el *flag* habilitado. Una vez terminado el procesamiento, se procede a enviar los resultados 6.30, 6.31 y 6.32. Cada resultado se muestra en una página que se genera, a partir del *response* que recibe y se acepta 6.29. Es decir, el peer remoto solo envia un **objeto JSON** con los resultados de lo que se solicitó. Luego, el Middleware delega el contenido realizando un forward hacia el puerto de conexión que existe con el Framework que utiliza la WebExtension NewsP2P.

Para este escenario, se utilizarón los contenedores 6.22 que han permitido generar Browsers por default con el Middleware instalado para que, luego, solo se tenga que instalar las WebExtension que necesitamos y realizar las pruebas de forma más sencilla.

Alternativa en modalidad de conexión y funcionalidad

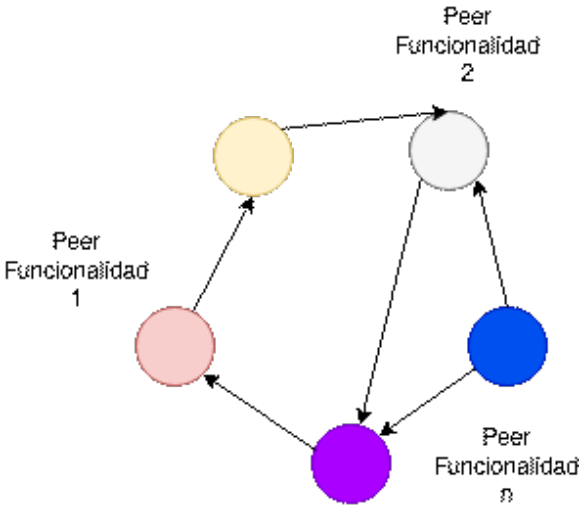


Figura 6.33: Modalidad de conexión alternativa de peers, donde la funcionalidad final racae en un peer determinado.

Como vemos en la figura 6.33, una alternativa de uso, por ejemplo, puede ser explorar el caso de envío de información a Peers específicos. La finalidad de está funcionalidad permite extender el uso de información más allá de la conexión de un Peer conectado con todos.

Es decir, imaginando un potencial de NewsP2P, donde las extracciones de tópicos de textos sean enviados a un usuario que conozca del tópico seleccionado. Al cual no está directamente conectado el usuario, sino que llegamos a él a partir de otro peer, con el objetivo que este último nos sirva como una pasarela de información hacia aquel idóneo que conozca del tópico en cuestión.

Por lo tanto, no solo sería posible acceder a información de otros usuarios que no estén conectados con uno mismo, sino que se podría pensar también en la posibilidad de que otros peers tengan más WebExtensions con otras funcionalidades. Y, por ejemplo, algunos podrían enviar el resultado en formato JSON, otros en CSV y otros juntar los resultados de ambos formatos y generar un gráfico. En ese caso, se llegaría a un caso de resolución en modo colaborativo, favoreciendo el lazo entre peers. Permitiendo al usuario no tener que instalar herramientas adicionales ni interactuar en otras páginas, ya que otros peers pueden ofrecer y compartir sus funcionalidades, a fin de tratar de solventar al peer que lo necesite, si es que está dentro de los límites de cada WebExtension.

A su vez, se puede pensar en un uso alternativo para la extracción de datos de páginas web como, por ejemplo, la minería de datos. En ese caso, cada uno de los peers individualmente enviará los datos a nodos más cercanos a fin de minimizar la latencia de red y mejorar la velocidad de tiempo de procesamiento. Por ejemplo, un peer cada n, sería el encargado en almacenar los resultado para que otros los procesen. De forma tal que se permite agregar una capa de seguridad en la información, durante el envío de mensajes. Para este caso, existe la posibilidad de generar mensajes seguros a través de una WebExtensions, como el caso MessageGuard[2].

Se puede concluir que es posible ir adaptando cada funcionalidad de WebExtension, así como la de *MessageGuard* para poder funcionar en modo P2P, y al mismo tiempo, complementar una funcionad existente, agregando una mejora como la posibilidad de funcionar como un **gateway de privacidad**.

[2]Es una WebExtension que permite la gestion de contenido seguro a traves de la red [24].

6.4. Ambiente de prueba y arquitectura

En está sección se comenta el desarrollo del laboratorio con el cual se han realizado las pruebas y las funcionalidades que necesito el proyecto. La construcción de WebExtensions que utilicen WebRTC por medio del browser a través de una arquitectura JSEP, para poder conectarse por medio de P2P requieren, al menos, tener en cuenta un desarrollo por capas. Para poder realizar pruebas semejante a un ambiente real y de usuario final, fue necesario utilizar tecnologías de virtualización y contenedores que nos permitan tener un ambiente controlado y, como principal protagonista, al Browser y su WebExtension.

Por lo tanto, desarrollar una arquitectura JSEP independiente del hardware y software, nos permitió aislar la capa de comunicación (acceso a redes LAN y WAN) que debe realizar una WebExtension en el browser. Dicha construcción también nos permitió utilizar la virtualización y contenedores para generar un ambiente controlado, que sea compatible con commodity hardware y, realizar un ambiente de pruebas personalizado, con la capacidad de realizar un número de escenarios posibles, en los que se pueda dar uso a recursos como cómputo y redes.

Los escenarios del laboratorio han requerido realizar un sistema operativo personalizado donde sea posible utilizar, como herramienta principal, un browser que pueda ser ejecutado como parte del sistema con recursos suficientes para garantizar el correcto funcionamiento. Con el propósito de ser ejecutado de forma automática, el browser con la WebExtension (Middleware) instalada, permite que un usuario o una máquina puedan realizar la interacción en el browser y utilizar la funcionalidad del Middleware. Para ello, se necesitó de un ingreso remoto. En nuestro caso, se optó por medio de un browser a través de un servicio de VNC.

En una primera aproximación, se realizaron pruebas con diferentes entornos de virtualización y se complemento con la utilización de contenedores, por medio de la plataforma Docker. Docker es una plataforma que permite construir y ejecutar contenedores a partir de imágenes, siendo una de las tantas alternativas de plataformas de containers que, además, es software libre. Sin entrar en muchos más detalles, la elección de utilizar contenedores se da porque permite un overhead menor, en cuanto a usos de recursos del sistema operativo y mejora el rendimiento frente a otros tipos de virtualización [25].

Para nuestro escenario, lo ideal es ahorrar y maximizar el uso del CPU,

disco, memoria y red. Aunque, para este último, se ha tenido que realizar una personalización de conectividad de redes, a fin de que cada contenedor pertenezca a una VLAN[3] determinada.

En este escenario, la tecnología de Virtualización junto a la paravirtualización que permite KVM en el Kernel de Linux fue de gran ayuda, ya que permitió conectar los contenedores por medio de redes virtualizadas. El desarrollo del sistema, que se ejecuta como un servicio en una imagen de Docker, usa como base la construcción de una imagen basada en el sistema operativo Ubuntu LTS minimal. Desde la misma, se puede personalizar y construir con los requerimientos de paquetes mínimos que permitan tener un sistema operativo Linux minimalista.

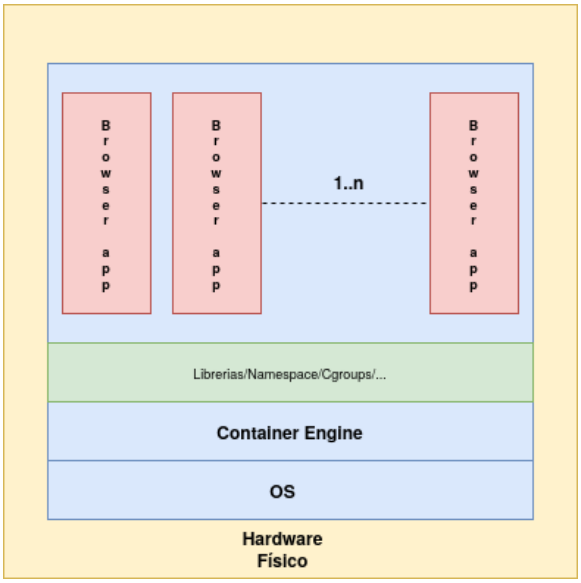


Figura 6.34: Aquitectura en capas.

[3]Virtual Private Network: Permite generar un identificador numérico a nivel de capa dos, permitiendo dividir los segmentos de red a nivel lógico.

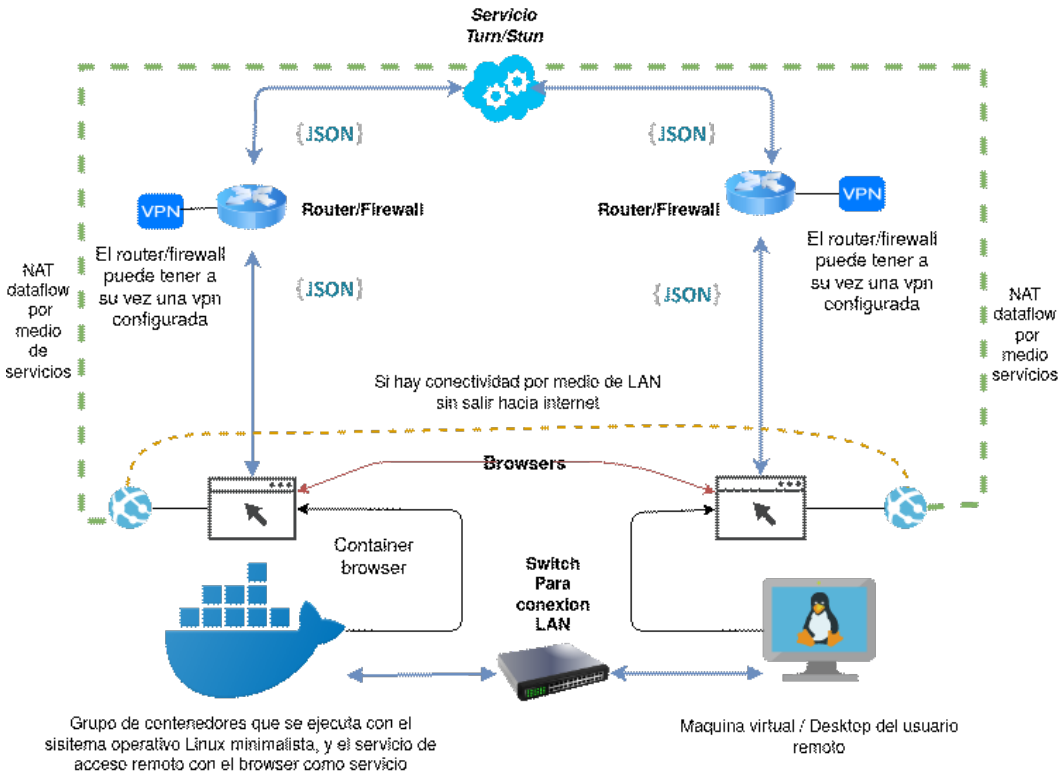


Figura 6.35: Ambiente de pruebas y desarrollo.

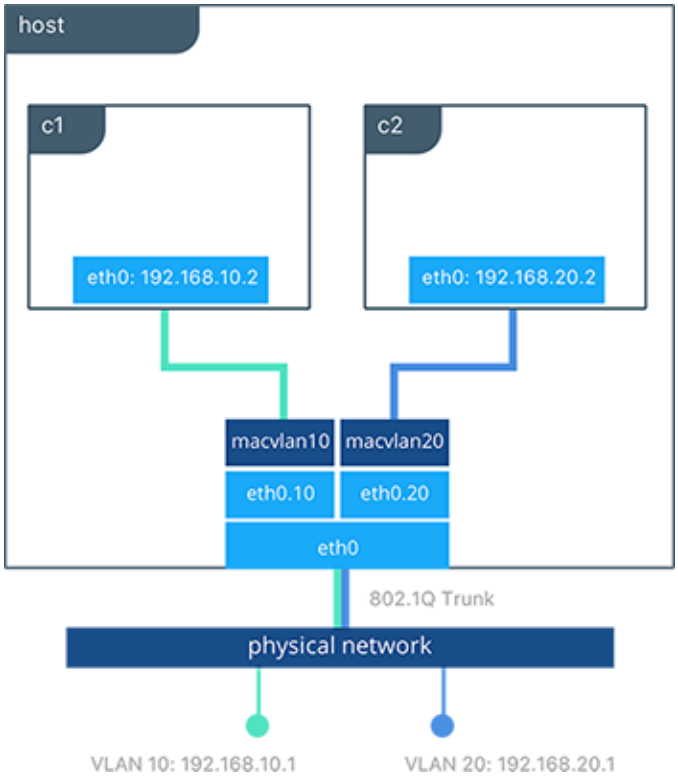


Figura 6.36: Modalidad Trunk-Macvlan utilizada en complemento con Docker.

En la figura 6.34 se pueden apreciar las capas, en lo que a contenedores se refiere y como el servicio de container, en este caso Docker, es el encargado de la gestión de cada aplicación que se ejecuta como un servicio dentro del container. Todas las aplicaciones que se ejecutan allí dentro interactúan con el mismo Kernel del host. Se debe recordar que al utilizar Docker también se nos permite exportar la imagen y, luego, poder ser ejecutada en otros ambientes o máquinas personales, sin la necesidad de disponer de tecnología de virtualización por hardware. Por lo tanto, permite utilizar las tecnologías existentes en el Kernel de Linux y como complemento, soporta containers. Otra de las ventajas es utilizar máquinas virtuales y, dentro de ellas, ejecutar Docker, a fin de ahorrar recursos y escenarios.

En la figura 6.35 podemos observar todas las capas en una pequeña infraestructura desplegada que nos permitió elaborar los diferentes escenarios de

prueba como, por ejemplo, la conectividad que necesita el Middleware para realizar las pruebas por medio de LAN. En ese caso, solo es necesario estar en el mismo segmento de red. Es decir, si tengo una máquina con dirección 192.168.10.10/24 puedo comunicarme con otra del mismo rango.

Mientras que, si estoy desde un contenedor en un segmento diferente, como por ejemplo, en otra red voy a necesitar pasar por un router/firewall y, para ello, se necesita de un servicio que nos permita saltar la restricción, en caso de ser necesario. Lo mismo ocurre en el caso de dos usuario conectados a través de internet.

Los escenarios como VPN fueron utilizados como pasarela de comunicación externa con el fin de probar diferentes escenarios de redes, con diferente latencia de conexión y agregando complejidad de rutas. Como se puede observar, el esquema de red utilizado en la figura 6.36 permitió la comunicación en la red LAN (a través de una modalidad de Trunk-Macvlan) y, a su vez, posibilitar la utilización de los router entre cada punto.

Capítulo 7

Conclusión y trabajo futuro

En conclusión al trabajo realizado, el Middleware es un híbrido, que funciona con la arquitectura JSEP. Esta arquitectura JSEP, genera una dependencia en los usuarios porque necesitan del servicio de señalización para poder conectarse hacia otros peers. Sin embargo, no está lejos de las soluciones de comunicación P2P, que se han implementado hasta el momento en otros proyectos como: Use Together 3.4.1, Pando 3.4.3, Dash 3.4.2, entre otros. Después de todo, es necesario tener una capa de comunicación que implemente la arquitectura JSEP. Además, se puede concluir que utilizar WebRTC para proveer P2P y para construir un Middleware que sea utilizado por medio de una WebExtension, es una ventaja. Debido a que permite utilizar la API de WebRTC que se encuentra implementada en el browser, simplificando la comunicación entre las capas. Con lo cual, tampoco se desvía del objetivo de P2P EUP porque permite al usuario utlizar el Middleware de forma transparente y simplificar el desarrollo de WebExtensions personalizadas, que quieran funcionar en forma P2P a través del uso del Framework. Y, otorga a los usuarios un medio para acercar al browser al concepto *computing in the edge*[1], potenciando una web descentralizada.

En vista de ello y lo que se ha comentado en la sección P2P y browsers 2.4, con este proyecto se espera lograr un aporte en cuanto al uso de tecnologías P2P y aportar al concepto de **Open Web Platform**, desde la pespectiva de aplicación práctica, después de todo, la posibidad de poder agregar una funcionlidad P2P al browser, nos permite tener aplicaciones a partir de We-

[1]En este contexto se tiene en cuenta al borde, como la distancia de acceso al recurso, es decir , la capacidad de llevar el contenido al usuario en lugar de ir a buscarlo en los servicios de red. Utilizando P2P.

bExtensions como partes de extensión de funcionalidad del browser. Donde es posible utilizarlas más allá de la red local. Además de esto, en este trabajo se ha podido exponer y evaluar el uso de WebExtension P2P, para animar la colaboración de peers, y favorecer a la creatividad colectiva; aportando y construyendo una web descentralizada.

Paralelamente, el uso del Middleware nos presenta otra ventaja, donde el usuario pueda construir su propia red de funcionalidades a través de peers. Siendo está red, una posible puerta hacia una web abierta y colaborativa que permita sortear los problemas de una red centralizada[2], Ya que al formar parte de una red P2P, el acceso a los recursos tiene más de una via de acceso y, este acceso, podría ser mejorado a través de los peers.

Como trabajo futuro, se espera poder realizar un mayor aprovechamiento de recursos, tanto de cómputo y almacenamiento. En donde se puedan apreciar mejoras en cuanto al uso de recursos compartidos. Por ejemplo, un tema interesante a explorar es la posibilidad del envío de streams de datos como video y archivos raw*Formato datos crudo, son los datos binarios del archivo.*. Respecto al recurso de almacenamiento, se podría realizar la factibilidad de proyectos como base de datos distribuidas, a través de P2P. Y en relación al cómputo, la posibilidad de aplicar Machine Learning distribuido, siendo este un tema interesante desde la perspectiva de la creación de modelos y ejecución de ellos, a través de pares. Al mismo tiempo, la navegación a través de internet también es un área que ha tenido diferentes alternativas de usabilidad. Por ejemplo, nativamente el browser presenta la posibilidad de utilizar proxys y hasta WebExtensions como VPN. Sin embargo, el uso de P2P podría extender la posibilidad de red paralela formada por peers. Por ejemplo, explorar la navegación colectiva de peers sería una modalidad de navegación potencial, utilizando los browsers de los usuarios cómo gateway de vpn. Permitiendo construir **puentes de navegación**; generando un sistema similar a lo que hoy vemos en la red Tor[3], solo que, en este caso, sería a través de browsers.

[2]Red centralizada, en cuanto al uso de la navegación web, asi como a realizar peticiones cliente-servidor. El acceso a recursos es a través del browser hacia el servidor, o grupo de servidores, por el echo de que pueden estar distribuidos a nivel geografico, pero no deja de ser de estar limitado al usuario por medio de un acceso centralizado a una dirección ip o varias(segun la redundancia del servicio.).

[3]Es un software que permite navegar en la red, a traves de otros nodos, favoreciendo la privacidad de la navegación. Este se puede distribuir por medio de un browser modificado y como un servicio.Link hacia Torproject.

Aquí se nombran, algunos trabajos a futuro potenciales:

- Construcción de aplicaciones P2P para realizar Maching Learning.
- Construcción de aplicaciones P2P para la transferencia de archivos sin tener en cuenta el formato.
- Construcción de aplicaciones P2P para permitir una navegación privada.
- Construcción de aplicaciones P2P para gestionar una base de datos distribuida.
- Construcción de una red overlay sobre la cual se puedan comunicar las aplicaciones. Este podria ser desde un protocolo P2P sobre WebExtension o un mecanismo de comunicación, del cual se pueda sacar provecho por medio de P2P.

Bibliografía

[1] W. C. Group., "Browser extensions. draft community group, https://browserext.github.io/browserext/, report 23 july 2017." https://browserext.github.io/browserext/.

[2] G. Bosetti, S. Firmenich, A. Fernandez, M. Winckler, and G. Rossi, "From search engines to augmented search services: An end-user development approach," in *Web Engineering* (J. Cabot, R. De Virgilio, and R. Torlone, eds.), (Cham), pp. 115–133, Springer International Publishing, 2017.

[3] D. Firmenich, S. Firmenich, G. Rossi, M. Winckler, and D. Distante, "User interface adaptation using web augmentation techniques: Towards a negotiated approach," in *Engineering the Web in the Big Data Era*, pp. 147–164, Springer International Publishing, 2015.

[4] G. Bosetti, S. Firmenich, S. E. Gordillo, G. Rossi, and M. Winckler, "An end user development approach for mobile web augmentation," *Mobile Information Systems*, vol. 2017, pp. 1–28, 2017.

[5] F. Daniel and M. Matera, *Mashups*. Springer Berlin Heidelberg, 2014.

[6] E. O. Burgueño, "Una vanguardia llamada collage." https://fido.palermo.edu/servicios_dyc/publicacionesdc/vista/detalle_articulo.php?id_libro=275&id_articulo=6837, 10 2010.

[7] E. Wohlstadter, P. Li, and B. Cannon, "Web service mashup middleware with partitioning of XML pipelines," in *2009 IEEE International Conference on Web Services*, IEEE, July 2009.

[8] O. Díaz and C. Arellano, "The augmented web," *ACM Transactions on the Web*, vol. 9, pp. 1–30, May 2015.

[9] I. Jacobs, J. Jaffe, and P. L. Hegaret, "How the open web platform is transforming industry," *IEEE Internet Computing*, vol. 16, pp. 82–86, Nov. 2012.

[10] R. Gonzalez, S. Firmenich, and G. Rossi, "A browser-based p2p architecture for collaborative end-user artifacts in the edge," in *End-User Development* (A. Malizia, S. Valtolina, A. Morch, A. Serrano, and A. Stratton, eds.), (Cham), pp. 234–238, Springer International Publishing, 2019.

[11] Gartner., "Gartner says global smartphone sales stalled in the fourth quarter of 2018." https://www.gartner.com/en/newsroom/.

[12] E. Lavoie, L. J. Hendren, F. Desprez, and M. Correia, "Pando: a volunteer computing platform for the web," *CoRR*, vol. abs/1803.08426, 2018.

[13] W. Hu, Z. Wang, and L. Sun, "Towards network-failure-tolerant content delivery for web content," *CoRR*, vol. abs/1607.01159, 2016.

[14] B. B. L. A. B. R. J. Rosenberg, A. Keranen, "Tcp candidates with interactive connectivity establishment." https://tools.ietf.org/html/rfc6544.

[15] J. Uberti., "Javascript session establishment protocol draft-ietf-rtcweb-jsep-25." https://tools.ietf.org/html/draft-ietf-rtcweb-jsep/.

[16] L. Lucas, H. Deleau, B. Battin, and J. Lehuraux, "USE together, a WebRTC-based solution for multi-user presence desktop," in *Lecture Notes in Computer Science*, pp. 228–235, Springer International Publishing, 2017.

[17] J. Bruneau-Queyreix, M. Lacaud, and D. Négru, "Increasing End-User's QoE with a Hybrid P2P/Multi-Server streaming solution based on dash.js and webRTC." working paper or preprint, Sept. 2017.

[18] M. Ogden, K. McKelvey, and C. f. S. Mathias Buus Madsen, "Dat - distributed dataset synchronization and versioning." Online Github, 01 2018.

[19] D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang., "Ed25519: high-speed high-security signatures." https://ed25519.cr.yp.to/.

[20] K. Jannes, B. Lagaisse, and W. Joosen, "The web browser as distributed application server: Towards decentralized web applications in the edge," in *Proceedings of the 2nd International Workshop on Edge Systems, Analytics and Networking*, EdgeSys '19, (New York, NY, USA), p. 7–11, Association for Computing Machinery, 2019.

[21] R. Gonzalez, S. Firmenich, A. Fernandez, G. Rossi, and D. Velez, "An approach to build p2p web extensions," in *Web Engineering* (M. Bielikova, T. Mikkonen, and C. Pautasso, eds.), (Cham), pp. 467–474, Springer International Publishing, 2020.

[22] M. Bachl, "Collaborative Home Network Troubleshooting," Master's thesis, Université Pierre & Marie Curie - Paris 6 ; INRIA, Sep 2016. page: 4-7.

[23] R. Pires, D. Goltzsche, S. Ben Mokhtar, S. Bouchenak, A. Boutet, P. Felber, R. Kapitza, M. Pasin, and V. Schiavoni, "Cyclosa: Decentralizing private web search through sgx-based browser extensions," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 467–477, 2018.

[24] S. Ruoti, J. Andersen, T. Monson, D. Zappala, and K. E. Seamons, "Messageguard: A browser-based platform for usable, content-based encryption research," *CoRR*, vol. abs/1510.08943, 2015.

[25] Z. Li, M. Kihl, Q. Lu, and J. A. Andersson, "Performance overhead comparison between hypervisor and container based virtualization," in *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pp. 955–962, IEEE, Mar. 2017.