Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Desarrollo de una aplicación VoIP para Kioscos en entornos hospitalarios

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Marc Borràs Miñana

Director: María Carmen Penadés Gramaje

Julio, 2015

Agradecimientos

En primer lugar quiero agradecer toda la ayuda prestada a mi directora de proyecto. Gracias por tus consejos y tu tiempo Mª Carmen.

A Jesús, ese gran amigo que siempre me ha aguantado durante estos años. Tienes mi respeto y mi admiración.

También a los cuatro pilares de mi vida.

Y finalmente a ella. Gracias por tu paciencia y tu alegría. *T'estime*.

Gracias a cada uno que ha gastado un segundo de su tiempo en ayudarme.

Resumen

Hoy en día los sistemas informáticos enfocados a sitios públicos con actividad constante y utilizados por usuarios inexpertos requieren ser muy intuitivos y claros, como apoyo a estos requisitos y a modo de asistente técnico, muchas veces se precisa de la interacción entre el cliente y un asistente experto que conozca el sistema.

A lo largo del proyecto trataremos de ofrecer una solución específica para la empresa Tecatel S.L, que posee un sistema de venta de tarjetas con chip para su posterior utilización en otras aplicaciones de la misma entidad. Se desarrollará una aplicación VoIP orientada a poner en contacto mediante una videoconferencia aquellos clientes que necesiten asistencia en el momento del uso del sistema con los asistentes de Tecatel. Esta aplicación será un módulo dentro del sistema principal, el cual es conocido como TeKio. Este módulo está consta de dos partes: la parte del cliente y la parte del asistente. Para el desarrollo utilizaremos el lenguaje de programación multiplataforma JAVA en una de sus últimas versiones. Con tal de poder establecer y definir las conexiones y comunicaciones entre las dos partes interesadas se utilizarán los protocolos *Session Initiation Protocol* (SIP) y *Session Description Protocol* (SDP) y debido a la naturaleza de éstos, se precisará de un servidor que los soporte. Y finalmente, para conseguir la persistencia de aquellos datos importantes utilizaremos bases de datos MySQL.

Palabras clave: VoIP, SIP, SDP, LibJitsi, Asistente.

Resum

Actualment els sistemes informàtics estan orientats a llocs públics amb activitat constant i usuaris inexperts, per tant requereixen ser molt intuïtius i clars. Com a ajuda a aquests requisits, actuant com a assistent tècnic, freqüentment es precisa de interacció entre el client i un assistent expert en el sistema.

Durant aquest projecte tractarem d'oferir una solució específica per a l'empresa Tecatel S.L, que posseeix un sistema de venta de targetes amb chip per al seu posterior us en altres sistemes de la mateixa entitat. Es desenvoluparà una aplicació VoIP orientada a ficar en contacte mitjançant una videoconferència aquells clients que precisen d'assistència alhora d'utilitzar el sistema, és a dir, desenvoluparem un mòdul auxiliar al sistema principal que és conegut amb el nom de TeKio. Aquest estarà separat en dos parts: la part client i la part assistent. Per al desenvolupament s'utilitzarà el llenguatge de programació multi plataforma JAVA en una de les seues últimes versions. Amb l'objectiu de poder establir y definir les connexions i el tipus de comunicació entre les parts interessades s'utilitzaran els protocols *Session Initiation Protocol* (SIP) i *Session Description Protocol* (SDP) i debut a la naturalesa d'aquests es precisarà d'un servidor que els suporte. I finalment per aconseguir la persistència de les dades més importants utilitzarem bases de dades MySQL.

Paraules clau: VoIP, SIP, SDP, LibJitsi, Asistent.

Abstract

Nowadays computer science programs are oriented to public places with constant activity and inexpert users, consequently they need to be very intuitive and clear, but sometimes users need a little help to use a system or to solve a problem.

Throughout this project we try to offer a specific solution for company Tecatel S.L., which has a chip card vending system, theese cards will be used in other company's applications. We will develop an application VoIP oriented to connecting clients who need help with assistants through a videoconference, in other words, an auxiliary module of principal system named TeKio. This system will be divided in two parts: client and assistant part. To develop we will use multi-platform JAVA programming language in one of last versions. To establish and define connections and the kind of communication between parts we will use two protocols: *Session Initiation Protocol* (SIP) and *Session Description Protocol* (SDP). Finally to achieve the persistence of most important data we will use a MySQL data base.

Keywords: VoIP, SIP, SDP, LibJitsi, Assistant.

Tabla de contenidos

Índice de ilustraciones

Índice de tablas

1. Introducción

En este capítulo presentaremos aquellos aspectos más importantes del contexto de nuestro proyecto, de este modo analizaremos los motivos que nos llevaron a realizarlo, así como los objetivos que éste debe cumplir. Y por último hablaremos de la estructura del propio documento para obtener una visión general de su organización.

1.1 Motivación

En la actualidad los sistemas informáticos que están orientados a sectores públicos con un gran potencial número de clientes inexpertos requieren ser muy intuitivos, claros y precisos. A pesar de que la mayoría de los productos son construidos cumpliendo estos tres principios, puede haber escenarios donde la satisfacción de estos no sea suficiente para facilitar su uso por parte de usuarios con poca o nula experiencia. Por otro lado, pero en menor número, existen escenarios realmente negativos que pueden darse durante el ciclo de uso de un producto. Teniendo como objetivo principal la satisfacción de los usuarios, dichos escenarios tienen un severo impacto en la imagen del producto. Por dichos motivos una gran cantidad de productos ofrecen diversos sistemas de asistencia o atención al usuario, con la intención de facilitar su uso, o bien contrarrestar los efectos negativos de escenarios adversos. Si "la calidad del producto y la **satisfacción del cliente** conforman el significado global de calidad" (H. Kan, 2002) entendemos que un sistema debe de poder superar estos escenarios negativos y facilitar lo máximo posible su uso por parte de los clientes. Otro punto que demuestra la importancia de la satisfacción del cliente es el valor que se le da en la norma ISO 9001 (norma de sistemas de gestión de calidad): una organización debe interesarse genuina y seriamente por la percepción que de sus productos o servicios. Por tanto cuando hablamos de la satisfacción de un cliente estamos hablando de un aspecto realmente importante de un producto.

Bien es sabido que toda aquella empresa que se considere de calidad debe disponer de atención al consumidor/usuario/cliente para resolver dudas, cuestiones y/o posibles problemas. Existen diversos servicios de asistencia basados en las distintas necesidades, construidas sobre distintas tecnologías. El mayor ejemplo de un servicio de atención al cliente es el telefónico, donde la comunicación es por voz utilizando tecnologías referentes a la comunicación por redes telefónicas. Sin ningún tipo de duda no podemos olvidarnos del servicio de atención realizado por recursos humanos de forma física. Otro servicio considerablemente usado es el correo electrónico, donde la comunicación es escrita y asíncrona, lo que implica un tiempo de espera por parte del cliente. Por otro lado tenemos distintos servicios de atención emergentes, basados en Internet, como el chat presente en grandes empresas como *Amazon*, *Dell*, *HTC, AT&T*... Servicios de atención síncronos o en tiempo real reducen de forma significativa el impacto negativo de un escenario dado e incluso pueden llegar a mejorar la impresión de un producto sobre un cliente.

Internet está incrementando su presencia en todos los aspectos de nuestra vida diaria. Con la llegada de los *Smartphones*, el concepto *Cloud*, el auge del *Internet de las Cosas*, los sistemas de seguridad conectados, servicios de contenido en *streaming*, sistemas

operativos en la nube… ha pasado a tener un papel realmente importante en nuestra sociedad. Por esto, sabemos que aquello que se implemente con tecnologías basadas en esta red tendrá soporte durante un buen periodo de tiempo.

Este proyecto se realiza en el marco de colaboración con una empresa, en concreto, nace motivado por la necesidad de ofrecer un servicio de asistencia para un sistema de venta de tarjetas de chip ya existente de la empresa Tecatel S.L, el software encargado de esta funcionalidad es denominado TeKio. TeKio se encuentra instalado en sistemas conocidos como Kioscos, ubicados en entornos hospitalarios, se encarga de realizar la venta, recarga y devolución de tarjetas con chip. Estas tarjetas son utilizadas para la reproducción de contenido multimedia en otros sistemas de la entidad.

A pesar que Tecatel dispone de trabajadores en todos los centros hospitalarios para ofrecer asistencia en el mismo centro, pueden existir momentos en los que los trabajadores no estén disponibles. Para evitar que los usuarios de los Kioscos con algún tipo de duda/cuestión/problema reciban una imagen negativa del sistema se decidió que la asistencia a ofrecer debería ser en tiempo real.

Por tanto podemos exponer que este proyecto nace de la necesidad de una empresa en ofrecer asistencia en tiempo real utilizando tecnologías actuales para un sistema ya existente. Este servicio de asistencia viene marcado claramente por la naturaleza del producto al que se le va a proveer apoyo. Los Kioscos son sistemas que están en espacios públicos, diseñados para ofrecer una interacción con los usuarios mediante una pantalla táctil. Debido a que el consumo de tiempo que requeriría el usuario para comunicarse mediante un chat es elevado comparado con una solución de comunicación por voz, se decidió que la solución debería soportar voz, y basándonos en lo anteriormente dicho se decidió utilizar los recursos ofrecidos por voz sobre IP (VoIP). *Voice over IP*, no es tan solo una forma de obtener llamadas relativamente gratis de larga distancia, el valor real de VoIP es que permite a la voz convertirse en nada más que otra aplicación basada en la red (Bryant, Madsen, & Van Meggelen, 2013).

Además y siguiendo en la filosofía de ofrecer una imagen de calidad de los productos se decidió incorporar la comunicación por video. En definitiva, se propuso proporcionar un servicio de atención al usuario basado en VoIP en tiempo real, reduciendo el tiempo de uso de los kioscos, mejorando así su tiempo útil y ofreciendo una imagen de profesionalidad.

1.2 Objetivos

El objetivo principal del proyecto es desarrollar una aplicación complementaria a TeKio capaz de poner en contacto visual y auditivo a un usuario del Kiosco con un asistente de la empresa. Así pues para alcanzar este objetivo deberemos cumplir los siguientes sub-objetivos:

1. Estudio del sistema TeKio para el correcto funcionamiento de la aplicación.
2. Determinar los requisitos de la aplicación.
3. Conocer las tecnologías a utilizar para el envío y recepción de audio y video.
4. Diseño de la aplicación.
5. Análisis y uso de los protocolos *Session Initiation Protocol* (SIP) y *Session Description Protocol* (SDP).

6. Desarrollo de una aplicación capaz de manejar las sesiones de comunicación, sesiones multimedia, los dispositivos y registrar aquellos datos importantes.

1.3 Estructura del documento

El presente TFG está estructurado en 7 capítulos y 3 anexos.

En el capítulo 2 podemos encontrar una breve explicación del contexto en el que se desarrolla en proyecto. Ofreceremos una breve visión de la empresa Tecatel, así como de sus sistemas y de diversos aspectos que influyeron en el desarrollo del TFG.

A lo largo del capítulo 3 presentaremos nuestra aplicación, destacando de manera general aquellos aspectos más importantes, como son sus requisitos funcionales y no funcionales, las distintas partes en la que se divide el sistema y la estructura de éstas. En definitiva se mostrarán aquellos aspectos importantes en los que posteriormente se profundizará.

Durante el capítulo 4 presentaremos se expondrá todo aquello relacionado con el modelado realizado para asegurar un correcto desarrollo. Se podrán observar diagramas de casos de uso, diagramas de clase, diagramas secuencia y diagramas de estado de aquellos aspectos más importantes del proyecto.

En el capítulo 5 encontraremos todo aquello referente a las tecnologías utilizadas destacando aquellos aspectos que tenga un mayor impacto. Hablaremos del entorno de desarrollo, los protocolos utilizados, el lenguaje de desarrollo y otros detalles.

El capítulo 6 ofrece información acerca del desarrollo de nuestra aplicación, proceso seguido, diseño de la interfaz, desarrollo con el lenguaje JAVA y pruebas realizadas.

El capítulo 7 presenta las conclusiones del TFG y posibles trabajos futuros.

Luego se presenta la bibliografía, y por último los anexos.

El anexo A contiene los aspectos más importantes para la configuración de la aplicación así como las instrucciones para la modificación del código fuente.

El anexo B contiene todos los casos de uso diseñados durante el proceso de modelado de nuestra aplicación, tanto los diseñados para la parte del cliente como la del asistente.

En el anexo C disponemos de la totalidad de los diagramas secuencia creados durante el proceso de modelado.

2. Estado del arte

En este capítulo ofreceremos información acerca de los Kioscos donde se implantará nuestro proyecto. Además también se presentará un breve análisis sobre tecnologías que pudieran ofrecer soluciones similares a la buscada y por último presentaremos datos relativos a las tecnologías utilizadas.

2.1 Kioscos y TeKio.

Como hemos comentado anteriormente la empresa ***Tecatel S.L*** dispone de sistemas de reproducción de contenido multimedia en *streaming*. Un subsistema de éste es el Kiosco; encargado de la venta de tarjetas de chip, que se utilizarán a modo de autorización en otro subsistema para la reproducción del contenido. No obstante, el Kiosco tan solo tendría las funciones de un ordenador a no ser por el *software* diseñado específicamente para él: **TeKio**. TeKio se encarga de ofrecer a los clientes la posibilidad de obtener tarjetas con distintos parámetros de autorización.

Actualmente los Kioscos funcionan con el sistema operativo Windows 7, no obstante existen versiones anteriores a las que se están actualizando que trabajan con Windows XP. Un Kiosco dispone tanto de billeteros como monederos para poder aceptar el pago por la compra de tarjetas. Y obviamente dispone de un emisor/lector de tarjetas con chip con el saldo seleccionado. Así complementando el primer párrafo podemos decir que TeKio es el software encargado de unificar y manejar todos los componentes con tal de ofrecer un sistema eficaz y transparente al usuario (ver Ilustración 1).

Ilustración 1: Entorno gráfico de Tekio.

Además como observamos en la Ilustración 1, es un sistema creado para un PC con pantalla táctil para facilitar la interacción con los usuarios. En la interfaz aparecen las tres funcionalidades realizables por un usuario: compra, recarga y devolución de tarjeta.

Destacar de TeKio que está escrito en *C Sharp* (C#) y utiliza servidores tanto WAMLP como LAMP para obtener distintos datos de configuración como pueden ser ofertas, precios, propiedades gráficas... A pesar que C# sea el lenguaje utilizado por la aplicación principal se decidió utilizar como lenguaje de desarrollo JAVA, debido a su soporte multiplataforma, asegurando el funcionamiento en caso de migraciones.

Las versiones de los Kioscos que trabajan en Windows 7 disponen de cámara, altavoz y micrófono por defecto, de este modo y partiendo de la base de que todos los kioscos tienen los mismos dispositivos de audio y video, en el desarrollo de nuestra aplicación centraremos nuestros esfuerzos en ofrecer una solución global para todos los kioscos futuros.

2.2 Posibles soluciones

La elección por parte de los empresarios de Tecatel de ofrecer una solución propia no fue trivial, antes de decidirse se realizaron estudios sobre distintas herramientas que tienen la capacidad de poner en contacto dos personas en tiempo real, que disponen de conexión a internet, una cámara, un micrófono, un altavoz y por supuesto una pantalla, como por ejemplo:

- Skype
- Hangout
- Jitsi
- WebEx

Debido a que el Kiosco es un sistema donde se limitan las funcionalidades que un usuario puede realizar, ninguna de las anteriores aplicaciones logró ofrecer una solución práctica para nuestro caso. Tanto Skype como Hangout son unas herramientas muy potentes con grandes funcionalidades, no obstante requieren que las dos partes de la comunicación tengan cierto conocimiento una de la otra, además ambas son realmente independientes y no hemos encontrado modo de que funcionen como complemento de TeKio. Por otro lado WebEx és otro producto descartado debido a la necesidad de un navegador para su funcionamiento. Se analizaron también otras tecnologías, no obstante, siempre nos encontrábamos con los mimos problemas, o bien era una solución cerrada sin posibles modificaciones, o bien eran difíciles de integrar en TeKio.

No obstante descubrimos Jitsi, que se trataba de una aplicación de código abierto escrita en JAVA. Esta aplicación ofrecía servicios de videoconferencia, VoIP y chat integrado, también era compatible con los protocolos SIP/SDP, y el protocolo de trasmisión de datos RTP, y poseía distintas funcionalidades a alto nivel que la enriquecía de forma notable. A pesar de esto, decidimos no adentrarnos en la modificación del código, por el gran coste temporal necesario para el entendimiento, sino que optamos por crear nuestra propia aplicación basándonos en una librería perteneciente a Jitsi conocida como **LibJitsi**.

Gracias a LibJitsi se nos abría un abanico de facilidades de cara al manejo de datos multimedia, tanto en su captura como en su codificación, envío y recepción de datos.

2.3 Tecnologías y protocolos utilizados.

El mayor punto a favor de la librería LibJitsi, para nuestras necesidades específicas y contando con las limitaciones temporales, era la capacidad de detectar dispositivos de captura de audio y video automáticamente, y sobretodo comunicarse con ellos con tal de obtener los datos digitalizados.

Se dispone de una cámara, un altavoz y un micrófono estandarizados para todos los kioscos futuros, un sistema operativo definido claramente, un entorno de ejecución cerrado y una librería de captura y envío de audio y video. No obstante, para poder realizar la comunicación precisábamos de un protocolo específico que nos ofreciera una solución real al problema del establecimiento de llamadas, el protocolo que decidimos utilizar fue el *Session Initiation Protocol* (SIP) para la negociación de sesiones y el *Session Descriprion Protocol* (SDP) para la descripción de estas.

Para entender esta elección debemos comentar ciertos aspectos de los sistemas reproductores de contenido en *streaming* así como el entorno donde trabajan. El primer aspecto importante es la capacidad de los reproductores de realizar una comunicación vía IP de audio entre dos sistemas utilizando el conjunto SIP/SDP. Más concretamente, el reproductor de *streaming* posee un micrófono y un altavoz incorporado que junto con el software, permiten al cliente comunicarse con un empleado de la empresa Tecatel S.L para realizar cualquier consulta o duda.

La comunicación mencionada en el párrafo anterior es posible gracias a un servidor SIP/SDP encargado de comunicar y localizar los distintos usuarios que existan en el sistema. Este servidor es conocido como Asterisk, en el apartado 5.4 recalcaremos los aspectos más importantes de éste. Obviamente podemos declarar que cada reproductor está asociado a un usuario presente en la base de datos del servidor Asterisk.

Por estos motivos, seleccionamos el conjunto SIP/SDP para el establecimiento de sesiones multimedia. Así disponíamos de diversos empleados de Tecatel con información extensa sobre los protocolos y el servidor Asterisk encargado de implementar su parte correspondiente.

De este modo al no encontrar una solución que se adecuara a las necesidades de la empresa **decidimos crear nuestra propia aplicación**. Una aplicación que fuera capaz de complementar a TeKio, establecer la conexión sin que el usuario dispusiera de datos de la parte con la que se quiere comunicar y sobretodo, capaz de comunicar el usuario con el asistente.

3. Propuesta de kiosco: Aplicación AsistenteVoIP

En este capítulo mostraremos los aspectos a destacar de nuestra aplicación denominada "AsistenteVoIP". Podremos ver los requisitos funcionales y no funcionales que se espera que cumpla, las partes en la que está dividida, la arquitectura de estas y cómo utiliza las tecnologías.

3.1 Requisitos funcionales y no funcionales.

Con el objetivo de ofrecer una solución adecuada para las necesidades y características del sistema ya existente, se decidió desarrollar una aplicación de comunicación. Para especificar correctamente los aspectos que debe cumplir de forma funcional o no, hablamos de requisitos funcionales y no funcionales respectivamente. Los requisitos funcionales describen las interacciones entre el sistema y el entorno independientemente de la implementación, por otro lado los requisitos no funcionales son aquellos aspectos que no están directamente relacionados con el comportamiento del sistema (rendimiento, recursos utilizados, seguridad y calidad son requisitos no funcionales de gran importancia) pero que son aspectos importantes (Bruegge & H. Dutoit, 2009).

Asistente VoIP debe ser capaz de cumplir los siguientes requisitos funcionales para garantizar su acoplamiento y su correcto funcionamiento:

- Registro de un usuario en el servidor para su localización.
- Enviar petición de llamada entre dos usuarios.
- Cancelar petición de llamada.
- Declinar petición de llamada.
- Establecimiento del canal de comunicación.
- Captura y codificación de video.
- Captura y codificación de audio.
- Recepción de video.
- Recepción de audio.
- Capacidad de pausar la llamada por parte del asistente.
- Reiniciar una llamada por parte del asistente.
- Observar el escritorio del kiosco.
- Observar la cámara del kiosco.
- Registro de las llamadas recibidas por el asistente.

Por otro lado, nuestra aplicación debe satisfacer los siguientes requisitos no funcionales (ya nombrados en el capítulo 2):

- Registro seguro de los usuarios en el servidor.
- Altamente configurable.
- Control total de las llamadas por parte del Asistente.

- Uso del protocolo SIP.
- Uso del protocolo SDP para el negocio de sesiones.
- Uso del servidor Asterisk.
- Desarrollada en el lenguaje JAVA.

De los requisitos no funcionales debemos destacar el protocolo SDP y la capacidad de la aplicación de ser altamente configurable. El SDP tiene una gran importancia en el envío y reproducción de contenido audiovisual, ya que mediante este las partes que quieren realizar el diálogo negocian los códecs que utilizarán durante la comunicación.

3.2 Arquitectura

Bass et al. (1998) define la arquitectura software de la siguiente manera:

> *"La arquitectura de software de un programa o sistema de computación es la estructura o estructuras del sistema, que comprende los componentes de software, las propiedades externamente visibles de esos componentes y las relaciones entre ellos".*

Por tanto, para nuestro sistema necesitamos de una arquitectura adecuada que se adecue a la definición anterior y nos permita encapsular los elementos de una forma eficaz y eficiente.

Como solución a este problema tenemos los patrones arquitectónicos, a lo largo de este documento presentaremos el patrón de arquitectura en tres capas. Se ha seleccionado este patrón entre muchos otros debido a la facilidad de independencia entre la lógica del sistema, la capa de persistencia y las interfaces gráficas posibles.

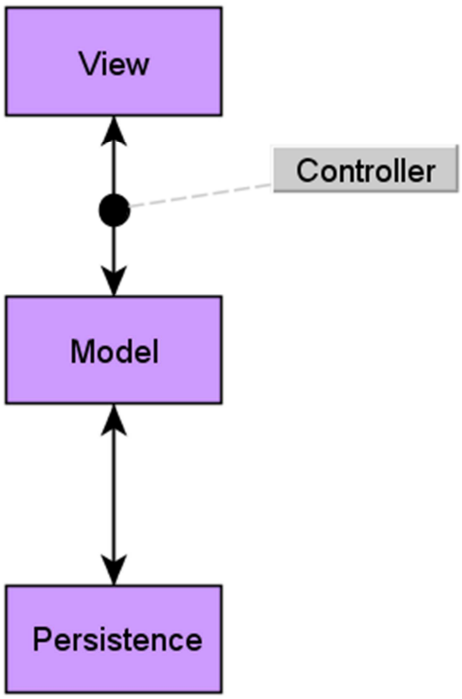


Ilustración 2: Arquitectura de la aplicación.

Como se observa en la Ilustración 2 disponemos de tres capas: *View*, *Model* y *Persistence*. La capa *View* maneja todo aquello relacionado con las interfaces de usuario (IGU), gracias a un controlador esta capa es independiente de la capa *Model* y por tanto todas aquellas acciones que realicen un cambio en la IGU deberán pasar por el controlador.

La capa *Model* se encarga de todo lo referente a la lógica de negocio de nuestra aplicación, mantiene una relación unidireccional con la capa *Persistence* ya que se encarga de comunicar a la persistencia aquellos datos que se quieren almacenar, en cambio la capa de persistencia no realiza ninguna acción sobre el modelo.

Como podemos deducir, la capa *Persistence* es la encargada de almacenar todos los datos recibidos desde *Model*. Como observamos en el gráfico no existe un controlador que se encargue de independizar la persistencia del modelo, esto es debido a la relación directa que existe entre los datos dinámicos de una llamada con los datos definitivos a ser almacenados.

Al tratarse de una aplicación que se ejecutará en dos sistemas distintos es necesario crear dos tipos de ejecutables, cada uno correspondiente a su lugar de ejecución. Así dispondremos de un ejecutable para la parte del Kiosco y otro para la parte asistente desde donde se atenderán las llamadas, no obstante, aún siendo ejecutables distintos ambos disponen de la misma arquitectura. Teniendo presente este detalle podemos pasar a presentar la arquitectura global del Asistente VoIP. Cómo observamos en la Ilustración 3 disponemos de una arquitectura cliente-servidor entre ambos ejecutables y nuestro servidor Asterisk, es decir, podemos entender nuestro sistema como una conjunto de servicios que son proporcionados por el servidor a los distintos clientes. Sin embargo, podemos comprender la arquitectura de dos formas distintas dependiendo del nivel de abstracción:

- En un nivel más bajo podemos observar el conjunto de nuestras partes y el servidor como una arquitectura cliente-servidor. Donde cada uno de los ejecutables actúa como cliente con el servidor SIP/SDP.
- A un nivel más alto podemos comprender la arquitectura global del sistema como una arquitectura cliente-servidor donde el Kiosco funciona como el consumidor de recursos y el PC Asistente como el proveedor.

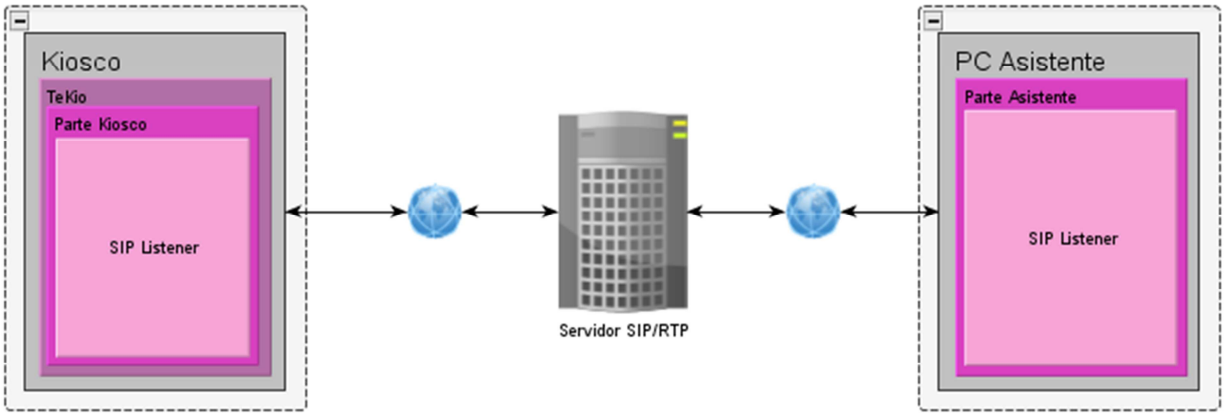


Ilustración 3: Arquitectura Cliente-Servidor entre las partes.

Por tanto, dependiendo del nivel de abstracción podemos comprender la arquitectura global de una forma o de otra. Y si descendemos todavía más bajo ya observaremos la estructura en tres capas de cada uno de los ejecutables presentes.

3.3 Uso de las tecnologías/protocolos

Tenemos las dos partes de nuestra aplicación en su propio entorno de ejecución y se comunican entre ellas a través de un servidor SIP. Para que ambas sean capaces de comunicarse cada una de ellas dispone de un SIP *Listener* encargado de procesar las peticiones y respuestas SIP. En el servidor SIP recae la responsabilidad de comunicar ambas partes para el establecimiento del negocio de las sesiones SIP/SDP, además también tendrá la responsabilidad de redirigir el tráfico de paquetes con contenido multimedia de una parte a la otra.

Además debemos destacar que cada parte es capaz de entender el contenido del mensaje de las peticiones y respuestas SIP, es decir, son capaces de interpretar el contenido SDP. De este modo no aseguramos que ambas partes puede saber qué tipo de contenido multimedia deben emitir y recibir. Para conseguir este objetivo se ha utilizado la librería JainSip, entraremos en más detalle en el apartado 6. No hace falta mencionar que ambas partes dispondrán de la librería LibJitsi, permitiendo así la recepción y envío de audio y video.

En este punto y habiendo presentado una idea general del protocolo SIP/SDP, la naturaleza de un servicio de atención al usuario y el propio objetivo de conseguir comunicar un usuario con un asistente de forma oral y visual podemos presentar el concepto de llamada. En el apartado 5 se entenderá mejor el concepto de llamada en el protocolo SIP, no obstante a alto nivel, una llamada la podemos definir como las acciones realizadas por un usuario para establecer comunicación con un asistente.

Por último comentar que JAVA ha sido el lenguaje seleccionado para el desarrollo de AsistenteVoIP por sus capacidades de ser independiente del sistema operativo y por los conocimientos previos del equipo de desarrollo. De este modo se dispone de dos ejecutables .jar cada uno correspondiente a la parte donde se ejecutará. El entorno de desarrollo para ha sido Eclipse debido a su amplio uso por parte de la comunidad JAVA y a su gran cantidad de funcionalidades para facilitar el desarrollo de código en JAVA.

4. Modelado conceptual

Para entender bien este capítulo, primero debemos conocer el lenguaje unificado de modelado UML. UML es un lenguaje de modelado de sistemas software respaldado por *Object Managment Group* (OMG) (OMG, 2014) diseñado para visualizar, especificar, construir y documentar software orientado a objetos (Booch, Jacobson, & Rumbaugh, 2006). Frente a los problemas y dificultades de diseñar software sólo con texto, UML añade un lenguaje visual facilitándose el diseño y mejorando claramente la verificación, validación y comunicación. Además aporta la capacidad de capturar el modelo de negocio definiendo los requisitos del sistema desde la perspectiva del usuario (Mishra, 2008). Con este lenguaje de modelado podemos observar la estructura y el diseño del software así como la forma en que todos los componentes del sistema se conocen. Creado concibiendo los aspectos fundamentales de los lenguajes orientados a objetos (OO), encaja perfectamente con las necesidades de este proyecto. UML dispone de 13 diagramas divididos en 6 categorías, no obstante nosotros nos centraremos en 4 de ellos: Diagrama de clases, Diagrama de secuencia, Casos de uso y Diagramas de estados.

4.1 Casos de uso

Los casos de uso se encargan de proporcionar uno o más escenarios donde se indica la forma en que un usuario debe interactuar con el sistema con tal de lograr un objetivo específico. Podemos definir un caso de uso como un conjunto de acciones que un sistema puede llevar a cabo gracias a la interacción con un actor (Jacobson, Christerson, Jonsson, & Gunnar, 1992). Con los diagramas de casos de uso logramos especificar el comportamiento de forma gráfica entre un usuario de la aplicación y el propio sistema, es decir con este diagrama representamos la relación directa entre los usuarios y los casos de uso posibles.

En el siguiente punto presentaremos los casos de uso modelados para nuestra aplicación. Hemos separado los casos de uso en dos modelos, uno para cada parte, de este modo disponemos de un modelo para la parte Asistente y otro para la parte Kiosco, no obstante siempre debemos mantener la idea de que ambos conjuntos conforman un único sistema.

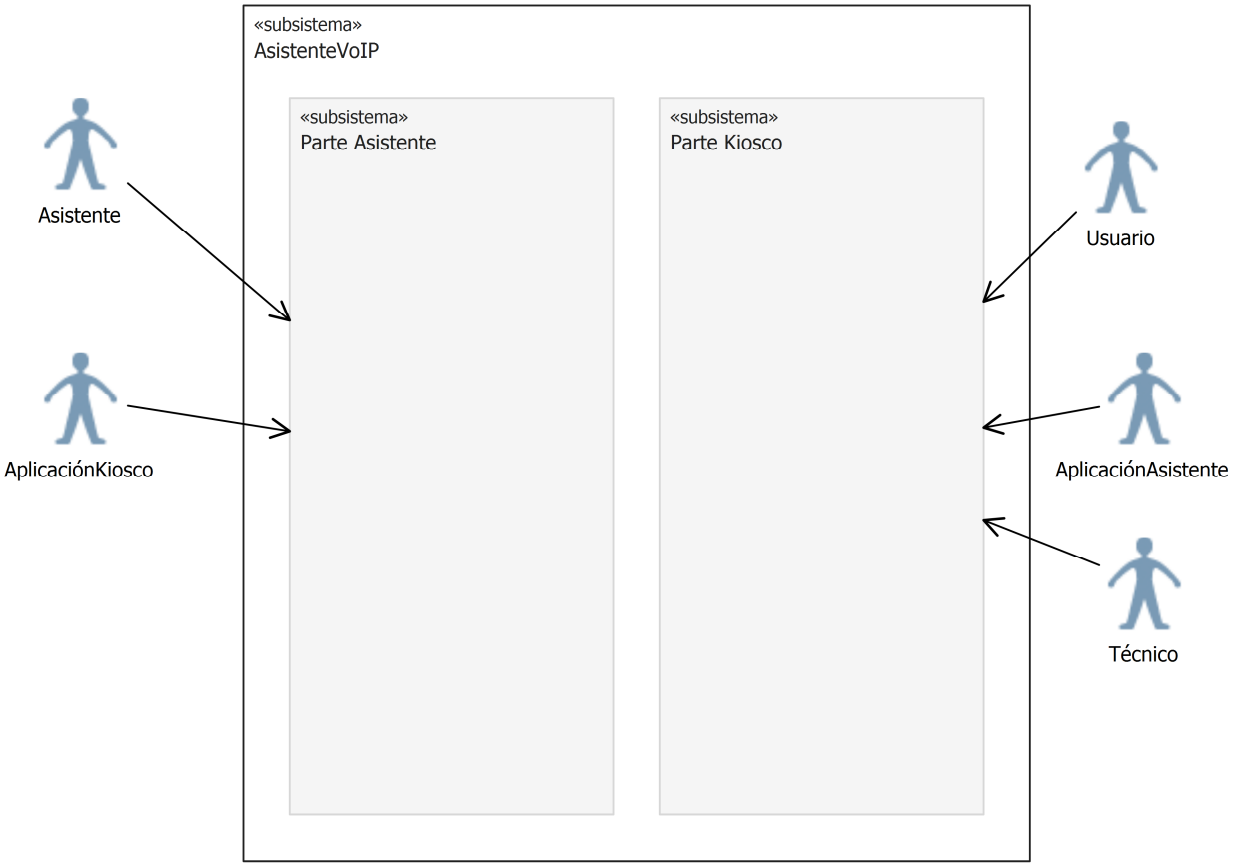


Ilustración 4: Modelo de contexto y diagrama inicial.

Los actores que interactúan con nuestro AsistenteVoIP, presentes en la Ilustración 4, están definidos en las siguientes tablas:

Actor	Asistente				
Descripción	Asistente de la empresa Tecatel S.L encargado de atender a los usuarios del kiosco.				
Características	Encargado del manejo casi total de las llamadas.				
Autor	Marc Borràs	**Fecha**	16/03/2015	**Versión**	2

Actor	Usuario				
Descripción	Usuarios del kiosco con la necesidad de contactar con el servicio de asistencia.				
Características	Tan solo son capaces de realizar peticiones de llamada.				
Autor	Marc Borràs	**Fecha**	16/03/2015	**Versión**	2

Actor	AplicaciónKiosco				
Descripción	Actor que representa las acciones que la aplicación del kiosco realiza sobre la parte asistente.				
Características	Todas aquellas acciones que tienen su origen en la aplicación del kiosco pero su efecto se realiza en la parte asistente son realizadas por este actor.				
Autor	Marc Borràs	**Fecha**	16/03/2015	**Versión**	2

Actor	AplicaciónAsistente				
Descripción	Actor que representa las acciones que la aplicación de la parte asistente realiza sobre la aplicación del kiosco.				
Características	Todas aquellas acciones que tienen su origen en la aplicación asistente pero su efecto se realiza en la parte del kiosco son realizadas por este actor.				
Autor	Marc Borràs	**Fecha**	16/03/2015	**Versión**	2

Actor	Técnico				
Descripción	Técnico de la empresa encargado de configurar la aplicación en la parte del kiosco.				
Características	Posee los conocimientos necesarios para realizar la correcta selección de los dispositivos.				
Autor	Marc Borràs	**Fecha**	16/03/2015	**Versión**	2

4.1.1 Parte asistente

En este apartado presentaremos los casos de uso más importantes relacionados directamente con la parte asistente de nuestra aplicación. En el anexo B se dispone de todos los casos de uso relacionados con la parte asistente. Los actores que interactúan con esta parte son dos: Asistente y AplicaciónKiosco. El primero actúa directamente en nuestra aplicación no obstante el segundo realiza acciones de forma indirecta a través de mensajes vía internet. Aún sin actuar directamente sobre la parte asistente, AplicaciónKiosco representa los mensajes generados en la parte Kiosco siendo un actor de gran importancia para el correcto funcionamiento del sistema.

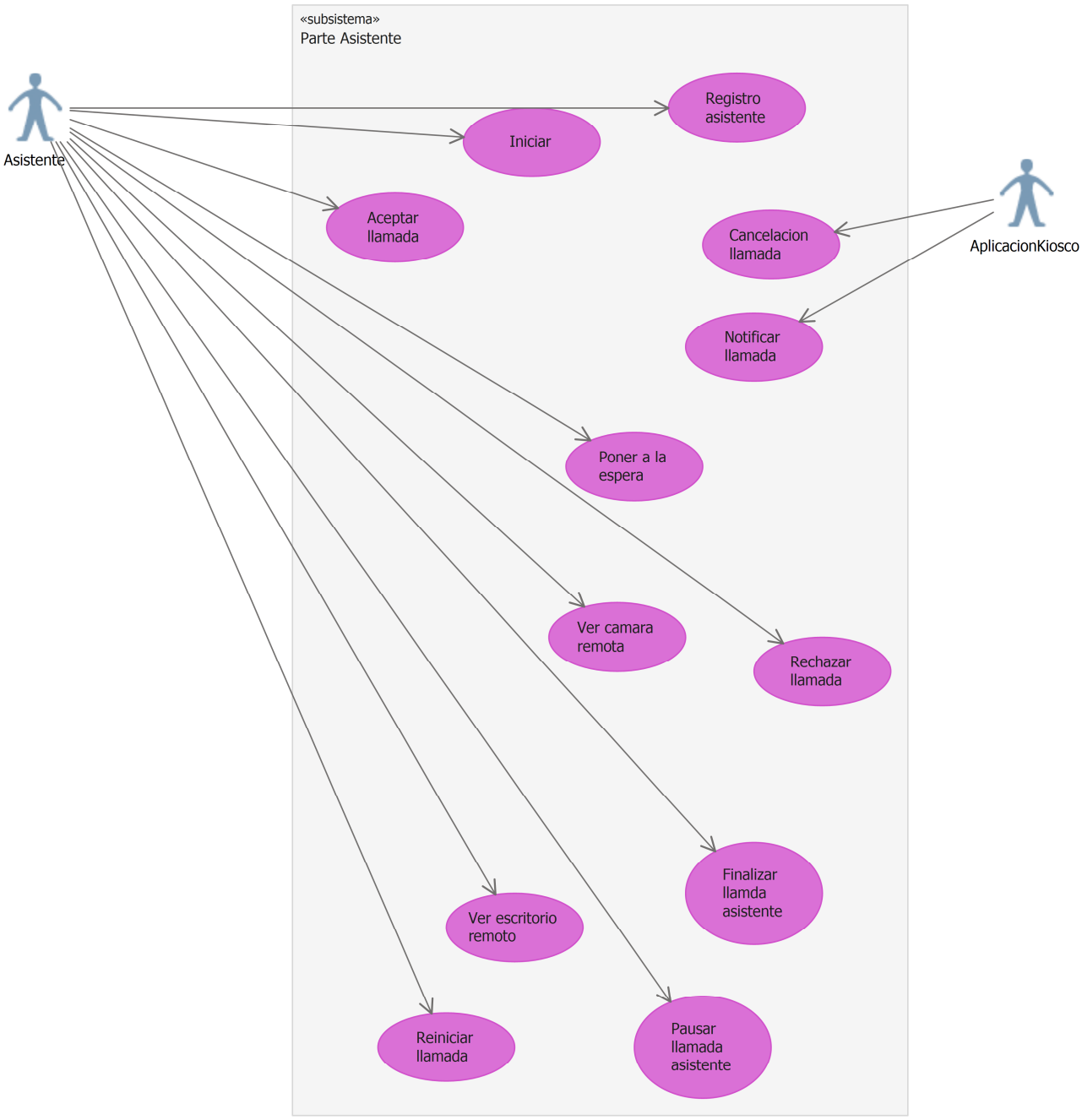


Ilustración 5: Casos de uso parte asistente.

Caso de uso	06- Rechazar llamada.
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	La aplicación deberá comportarse tal como se describe en el siguiente caso de uso cuando el asistente rechace una llamada.
Precondiciones	Se ha realizado el caso de uso 01 y 16.
Postcondiciones	-
Incluye	-
Extiende	-
Hereda de	-

Flujo de Eventos	
Usuario	**Sistema**
1. El asistente decide rechazar una llamada entrante.	2. Envía una respuesta de rechazo al servidor SIP.
	3. Oculta la ventana de notificación de la llamada.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. Si en 2 no se puede contactar con el servidor después de varios intentos, la parte kiosco quedará sin notificar el rechazo de la petición. Se sigue el flujo normal para el rechazo de una llamada.
Comentarios	-

Caso de uso	07- Aceptar llamada.
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	La aplicación deberá comportarse tal como se describe en el siguiente caso de uso cuando el asistente acepte una llamada.
Precondiciones	Se ha realizado el caso de uso 01 y 16.
Postcondiciones	Se abre una ventana donde se observan las imágenes recibidas y enviadas. También se empieza a reproducir el audio recibido y a enviar el capturado.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El asistente acepta una llamada entrante.	2. Se leen los datos de comunicación para el establecimiento de una sesión SIP y los datos necesarios para establecer una sesión multimedia (SDP).
	3. Se envía una respuesta de aceptación al servidor SIP con los datos necesarios para la comunicación.
	4. Se inicia el envío y recepción de audio y video.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. En caso que en 3 no se pueda contactar con el servidor SIP no se inicia el envío y recepción de audio y se le notifica al asistente.
Comentarios	-

Caso de uso	10- Finalizar llamada asistente.
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	El asistente decide finalizar la comunicación con un Kiosco determinado.
Precondiciones	Se ha realizado el caso de uso 07.
Postcondiciones	Ya no existe ningún tipo de comunicación con el Kiosco llamante.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El decide finalizar una llamada determinada.	2. Se le comunica al servidor la decisión de finalizar la llamada, mediante los mensajes pertinentes.
	3. El sistema finaliza el envío y recepción de audio y video.
Extensiones síncronas	
	1. En 3 para el caso en que la llamada este pausada tan solo se finaliza la recepción de audio.
Extensiones asíncronas	
	1. En el paso 2 la aplicación no consigue contactar con el servidor, se finaliza la llamada en parte asistente y termina el caso de uso.
Comentarios	-

Caso de uso	15- Iniciar.
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	El asistente inicia la parte Asistente y el sistema carga los datos necesarios del fichero de configuración.
Precondiciones	-
Postcondiciones	-
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El asistente inicia la aplicación.	2. El sistema lee todos los datos referentes a conexiones con servidores (SIP y MySQL), los usuarios SIP relacionados con el asistente y datos referentes a la transmisión de contenido multimedia.

	3. El sistema comprueba la estructura correcta de los datos. Si los datos son correctos se realiza el caso de uso 08.
Extensiones síncronas	
-	1. En 2 si no se encuentra el fichero se notifica al usuario y finaliza el sistema.
-	2. En 3 si los datos no poseen una estructura correcta el sistema lo notifica al asistente y finaliza.
Extensiones asíncronas	
-	-
Comentarios	-

4.1.2 Parte kiosco

Para completar el apartado 4.1 presentaremos los casos más de uso más importantes directamente relacionados con la parte Kiosco de nuestra aplicación (en el anexo B se pueden consultar todos los casos de uso de la parte Kiosco). Así como para la parte asistente se dispone de distintos actores, aquí debemos diferenciar entre tres actores: Técnico, Usuario y AplicaciónAsistente. Los dos primeros tienen influencia directa, no obstante el último realiza sus acciones en el sistema mediante mensajes a través de internet. Como se puede deducir el usuario AplicaciónAsistente y el AplicaciónKiosco son complementarios y contrarios, dicho de otro modo, AplicaciónAsistente representa las acciones que se realizan en la parte Kiosco mediante los mensajes realizados en la parte Asistente. Por tanto, así como AplicaciónKiosco, este actor posee una gran influencia en el correcto funcionamiento del sistema.

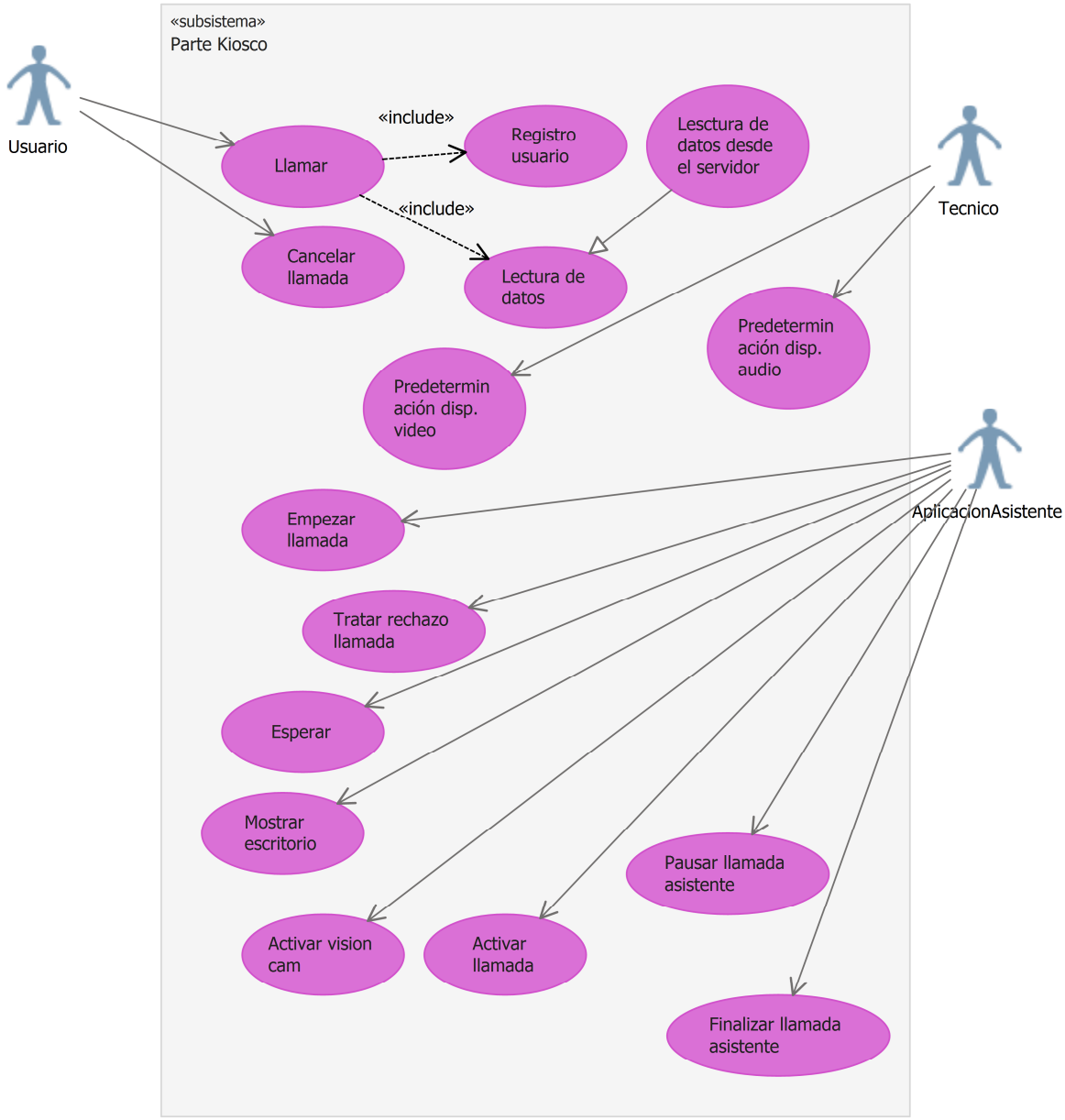


Ilustración 6: Casos de uso parte kiosco.

Caso de uso	01- Llamar.
Actor	Usuario.
Entorno de ejecución	Parte Kiosco.
Resumen	Un usuario decide ponerse en contacto con un asistente.
Precondiciones	-
Postcondiciones	-
Incluye	02 y 03.
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El Usuario pulsa sobre el botón de asistencia remota.	2. Seguidamente se realiza el caso de uso 03 y posteriormente el 02.

	3. Se envía un mensaje al servidor indicando la petición llamada.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. En 3 si no se puede contactar con el servidor el sistema se lo notifica al usuario y finaliza la aplicación.
	2. En 3 si el servidor responde que el asistente no está disponible el sistema se lo notifica al usuario y finaliza la aplicación.
Comentarios	-

Caso de uso	02- Registro usuario.
Actor	Usuario.
Entorno de ejecución	Parte Kiosco.
Resumen	El sistema debe registrar el usuario en el servidor SIP a fin de identificar el Kiosco llamante y permitir realizar llamadas.
Precondiciones	Se ha realizado el caso de uso 03 y el caso 01 está en proceso.
Postcondiciones	-
Incluye	-
Extiende	01.
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
	1. Envía los mensajes necesarios para el registro SIP en el servidor.
	2. El sistema responde aceptando el registro del usuario emparejado con el kiosco.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. En 1 no se puede establecer conexión con el servidor, el sistema lo comunica al cliente y finaliza.
	2. En 2 el servidor responde que el usuario no ha sido registrado con éxito, por tanto el registro falla, se le notifica al usuario y finaliza la aplicación.
Comentarios	-

Caso de uso	05- Cancelar llamada.
Actor	Usuario.
Entorno de ejecución	Parte Kiosco.
Resumen	Después de realizar una petición de llamada el usuario decide cancelar la petición de comunicación.

Precondiciones	Se ha realizado el caso de uso 01 y aún no se ha realizado 07.
Postcondiciones	No se establece comunicación con el asistente.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El usuario decide no establecer comunicación con el asistente.	2. Se envía al servidor la petición de cancelación de llamada y el sistema finaliza.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. En 2 la conexión con el servidor no se realiza de forma satisfactoria, se notifica el error al usuario y finaliza el sistema.
Comentarios	-

Caso de uso	18- Empezar llamada.
Actor	AplicacionAsistente.
Entorno de ejecución	Parte Kiosco.
Resumen	El asistente acepta una llamada y la parte Kiosco realiza las acciones necesarias para la comunicación.
Precondiciones	Se ha realizado el caso de uso 07.
Postcondiciones	Se abre una ventana donde se observan las imágenes recibidas y enviadas. También se empieza a reproducir el audio recibido y a enviar el capturado.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. AplicaciónAsistente envía un mensaje notificando la aceptación de la llamada.	2. La parte Kiosco recibe el mensaje y notifica la recepción de la aceptación mediante un mensaje.
	3. Se guardan los datos importantes para la transmisión y recepción de audio y video a partir del mensaje recibido.
	4. Se inicia el envío y recepción de audio y video.
Extensiones síncronas	
-	-
Extensiones asíncronas	
-	-
Comentarios	El mensaje enviado es el generado en el caso de uso 07.

Caso de uso	27- Finalizar llamada asistente.
Actor	AplicaciónAsistente.
Entorno de ejecución	Parte Kiosco.
Resumen	El asistente finaliza una llamada y la parte Kisoco finaliza.
Precondiciones	Se ha realizado el caso de uso 18.
Postcondiciones	-
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. AplicacionAsistente envía un mensaje informando de la finalización de la llamada.	2. El sistema recibe el mensaje y confirma la recepción de este mediante otro mensaje.
	3. Termina el envío y recepción multimedia y finaliza el sistema.
Extensiones síncronas	
-	-
Extensiones asíncronas	
-	-
Comentarios	-

4.2 Diagrama de clases

Como complemento a los diagramas de casos de uso tenemos el diagrama de clases, perteneciente también al lenguaje de modelado UML. Mientras los casos de uso muestran las interacciones de un sistema con los actores, los diagramas de clase son diagramas estructurales que muestran la estructura estática del sistema, las clases del sistema y las relaciones entre estas. Se entienden como clases las abstracciones que representan una estructura, y su comportamiento, de un grupo de objetos. Mediante la instanciación de las clases tenemos lo objetos que son creados, modificados y destruidos a lo largo de la ejecución del sistema (Bruegge & H. Dutoit, 2009).

Los diagramas de clases y los de casos de uso forman una herramienta ampliamente utilizada en el modelado de aplicaciones.

A lo largo de este punto 4.2 presentaremos los diagramas de clase de las dos partes presentes en nuestra aplicación junto con una explicación de cada uno. De igual modo que para los diagramas de casos de uso debemos mantener en mente que ambos diagramas forman parte de una misma aplicación.

4.2.1 Parte asistente

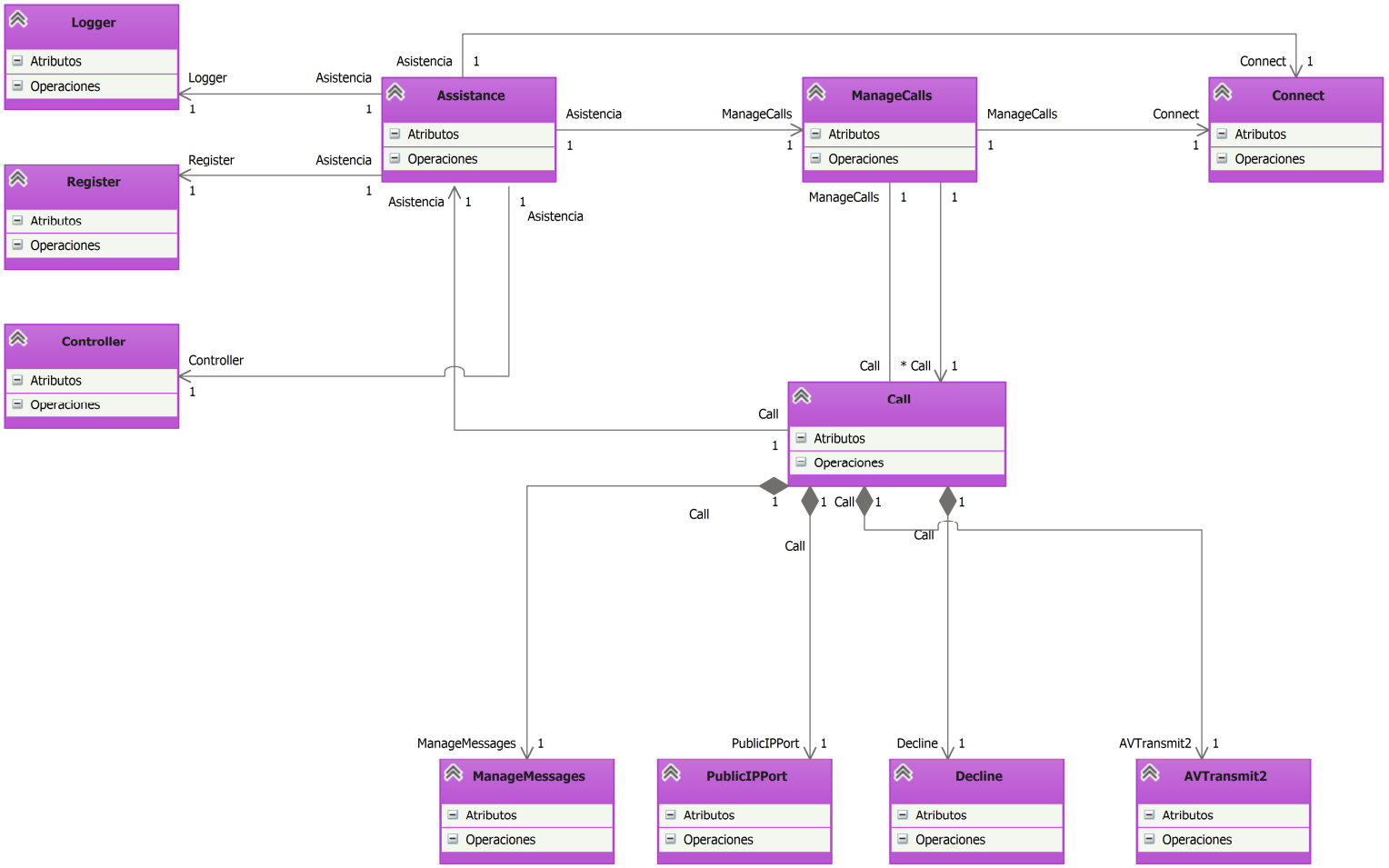


Ilustración 7: Diagrama de clases de la parte asistente.

Como podemos observar en la Ilustración 7 se trata de un diagrama bastante completo, del que explicaremos las clases más importantes y sus relaciones.
La clase *Assistance* inicia la aplicación, además se encarga de leer todos aquellos datos relacionados con el registro del asistente en el servidor SIP guardados en un fichero. También tiene la responsabilidad de crear todos aquellos objetos que sean necesarios para la comunicación con el servidor y será el punto de recepción de las peticiones y respuestas SIP, es decir, será el SIP *Listerner* que hemos comentado anteriormente.

Posee una instancia de la clase *Logger* que se encarga de mantener un registro de las llamadas recibidas (fecha, hospital y kiosco) almacenando los datos en un fichero de texto a modo de complemento de la clase *Connect*.

Otra clase que posee es *Register*. Ésta es la encargada de crear las peticiones de registro (necesarias para la comunicación multimedia) que se enviarán al servidor, así como de procesar las respuestas recibidas relacionadas con el registro de un usuario SIP y de crear las cabeceras específicas para este.

La clase *Connect* se encarga de guardar en una base de datos relacional todo aquello que se decidió que era importante respecto a una llamada: hospital, kiosco, fecha inicio llamada, fecha fin llamada, si ha sido atendida, usuario origen y estado de la llamada. El estado de la llamada se define basándose en la siguiente tabla:

Estado	Valor
0	Se ha recibido una nueva llamada. Llamada pendiente de aceptarse.
1	Llamada aceptada.
2	Llamada en espera.
3	Llamada cancelada por el usuario.
4	Llamada finalizada.
5	Llamada activa.
6	Llamada rechazada por el asistente.

Tabla 1: Tabla estados de una llamada.

Basándonos en la información almacenada de cada una de las llamadas se pueden realizar distintos estudios para mejorar nuestro sistema y empresa.

La clase *ManageCalls* es de gran importancia ya que es la encargada de gestionar todas las llamadas recibidas y actualizar todos los datos referentes a éstas gracias a la instancia de *Connect* que posee. Otra clase importante es *Call,* que contiene todos los datos importantes de las llamadas recibidas. Podemos entender esta última como la representación de cada petición de comunicación enviada por parte del usuario del kiosco. La clase *Call* está formada por estas otras clases:

- *Decline*: tiene la responsabilidad de crear aquellos objetos necesarios para el rechazo de las llamadas recibidas.
- *PublicIpPort*: Se encarga de obtener la IP pública de nuestra red local y el equivalente a ciertos puertos locales una vez pasado el *Network Address Translation* (NAT).
- *AVTransmit2*: Si la llamada fuera aceptada, esta clase se encargaría de efectuar el envío y recepción de audio.
- *ManageMessages*: Se encarga de manejar los mensajes SIP recibidos.

La clase *Controller* se encarga de comunicar la capa de presentación con la de lógica, a pesar de ser un controlador tiene una gran importancia en la lógica de nuestra aplicación. Gracias a esta se mejora la independencia entre capas, la portabilidad y el mantenimiento de la aplicación.

4.2.2 Parte kiosco

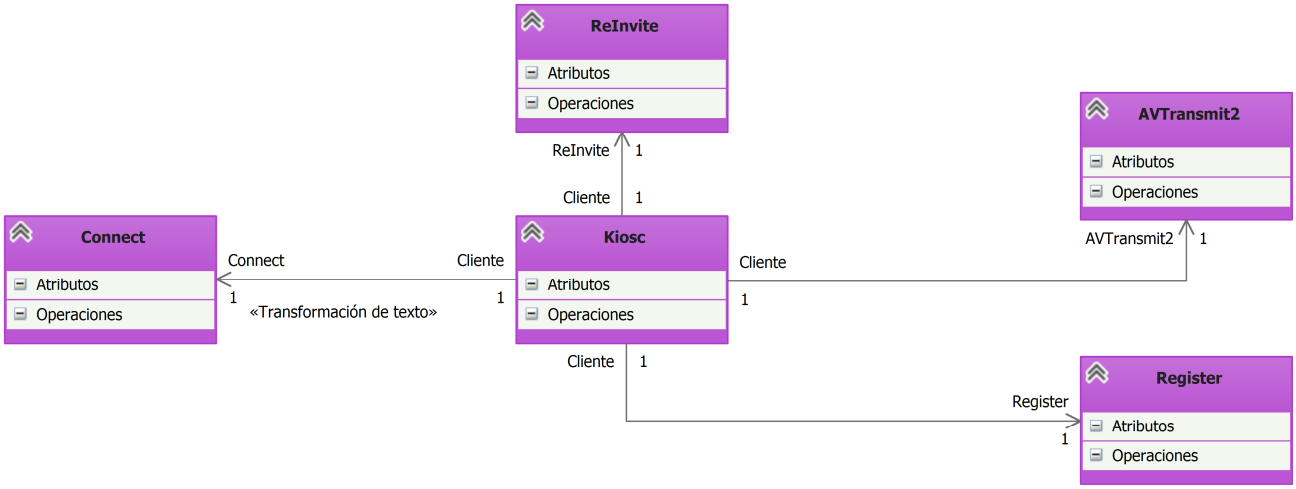


Ilustración 8: Diagrama de clases de la parte Kiosco.

La parte kiosco es mucho más sencilla que la parte asistente, debido a que la mayoría de las funcionalidades disponibles en el sistema residen en la parte Asistente. Como en el apartado anterior procedemos ahora a realizar una breve explicación de cada clase y relación que aparece en el diagrama.

La parte cliente se inicia desde la clase *Kiosc* y se encarga de crear todos aquellos objetos necesarios para la comunicación SIP con el servidor al igual que la clase *Assistance*. Además, será el punto de recepción de peticiones y respuestas implementando el SIP *Listener* presentado en el apartado 3.2.

AVTranmist2 tiene la misma funcionalidad para ambas partes de nuestro sistema, por tanto, se trata de una clase replicada en cada parte e independizada en cierta forma del contexto con el fin de ofrecer sus capacidades y competencias a los dos ejecutables. La misma filosofía de diseño se ha seguido para la clase *Register*, de este modo también disponemos de sus funcionalidades en cada parte.

A diferencia del diagrama anterior, en este, la función de la clase *Connect* es la de cargar los datos necesarios para la comunicación vía SIP, obteniéndolos de una base de datos o de un archivo de configuración. También se dispone de la clase *ReInvite* encargada de enviar mensajes *INVITE* al servidor con tal de mantener la comunicación SIP abierta.

4.3 Diagramas secuencia

Siguiendo con el modelado de nuestra aplicación presentamos los diagramas secuencia modelados. Estos diagramas se utilizan principalmente para representar las interacciones entre objetos siguiendo el orden en que estas ocurren. Los diagramas secuencia nacen como la elevación de los casos de uso proveyendo un nivel más formal de refinamiento (IBM, 2015). Normalmente los casos de uso son refinados en uno o más diagramas secuencia.

4.3.1 Parte asistente

En este punto mostraremos los diagramas que se consideren más importantes e informativos para obtener una idea de las interacciones que destacan del sistema. El resto de diagramas creados se encuentran en el anexo C.

Como hemos comentado, cada caso de uso pude ser refinado en uno o más diagramas de clase. Por tanto podemos encontrar distintos diagramas en los que aparecen diagramas auxiliares, estos diagramas también los encontraremos en el anexo C.

4.3.1.1 Diagramas principales

- 06-Rechazar llamada. Identificador del diagrama: **sd RechazarLlamada**.

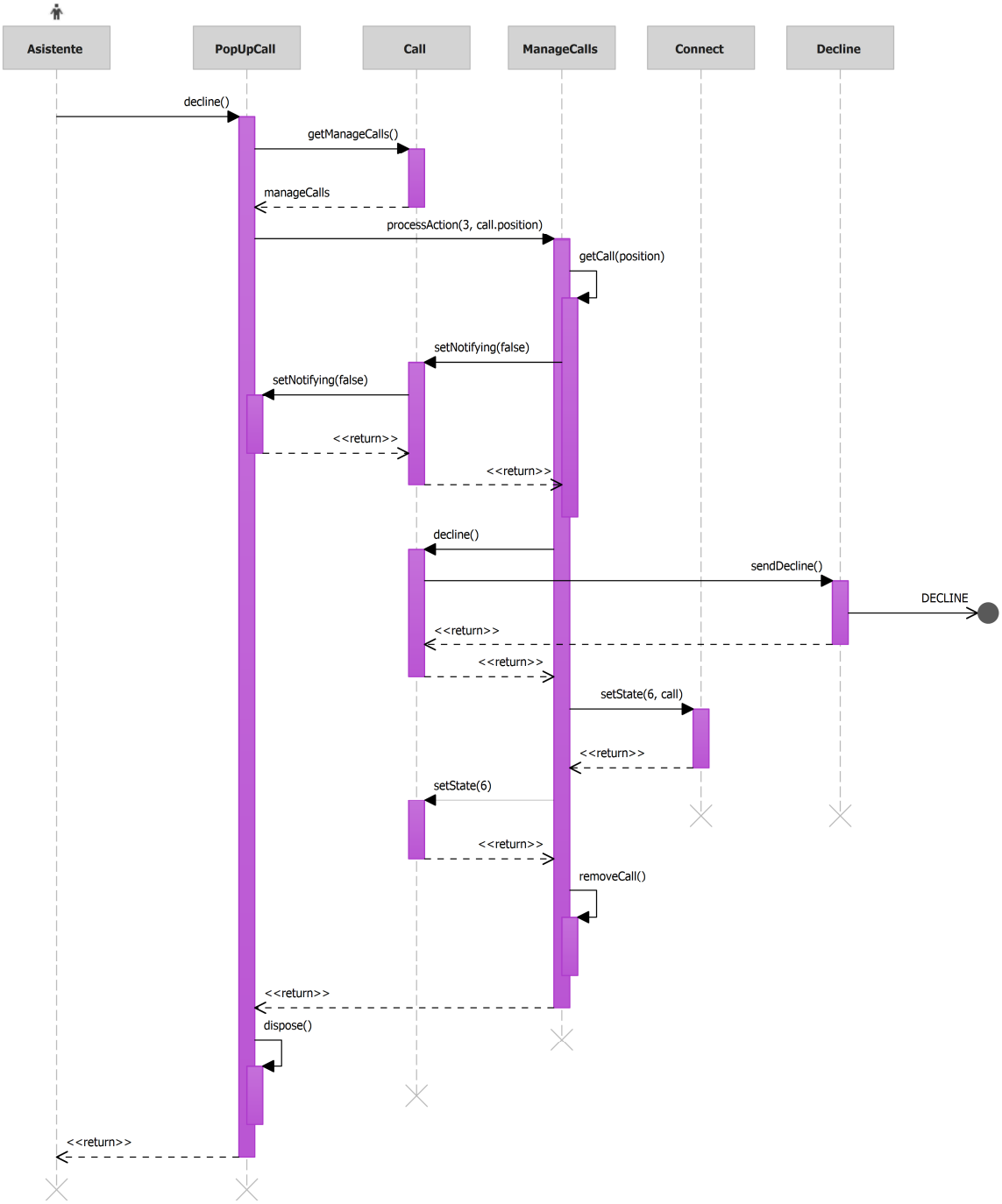


Ilustración 9: sd RechazarLlamada.

- 07-Aceptar llamada. Identificador del diagrama: **sd AceptarLlamada**.

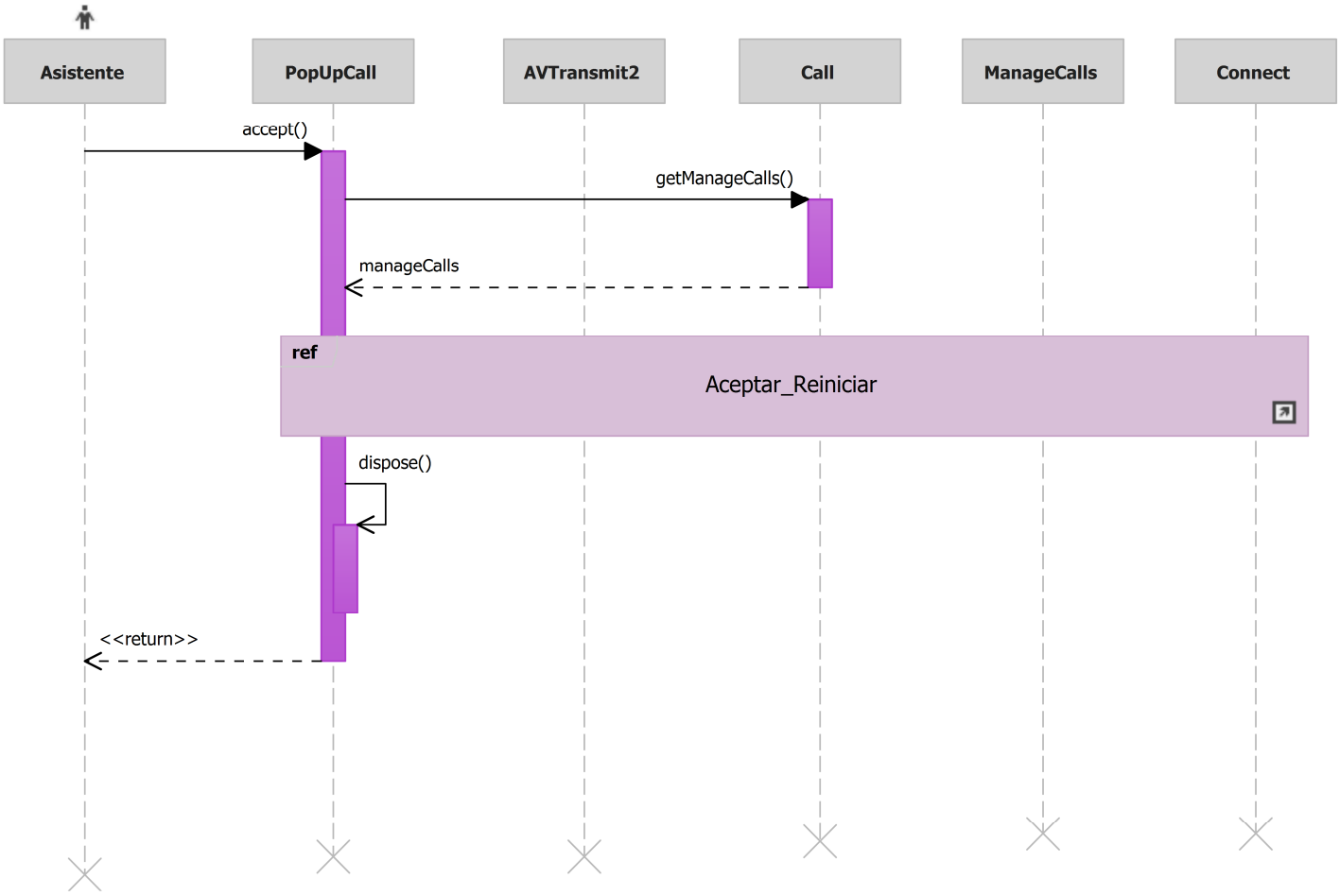


Ilustración 10: sd AceptarLlamada.

- 10-Finalizar llamada asistente. Identificador del diagrama: **sd FinalizarLlamadaAsistente**.

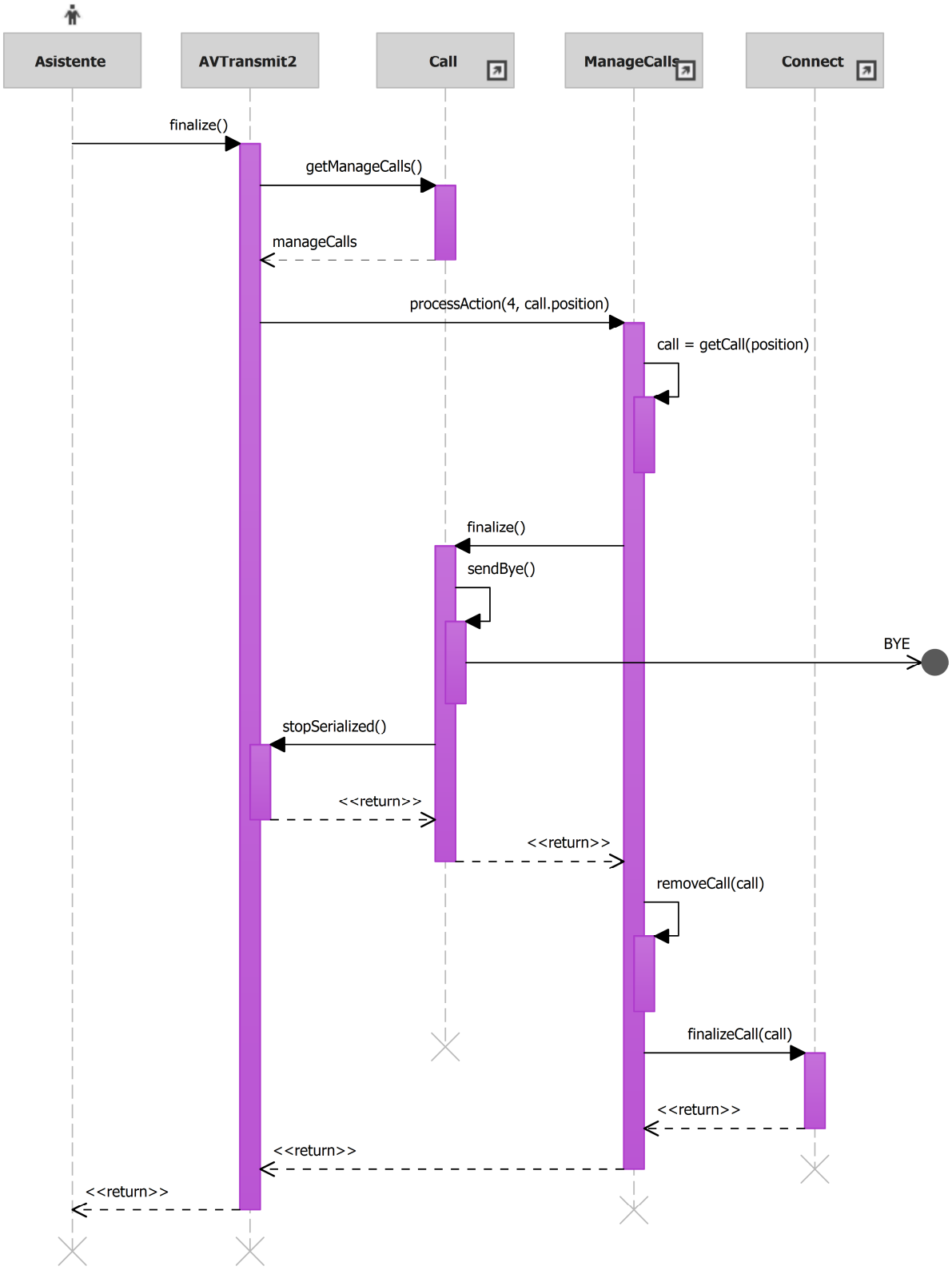


Ilustración 11: sd FinalizarLlamadaAsistente.

- 15 – Iniciar. Identificador del diagrama: **sd Iniciar**.

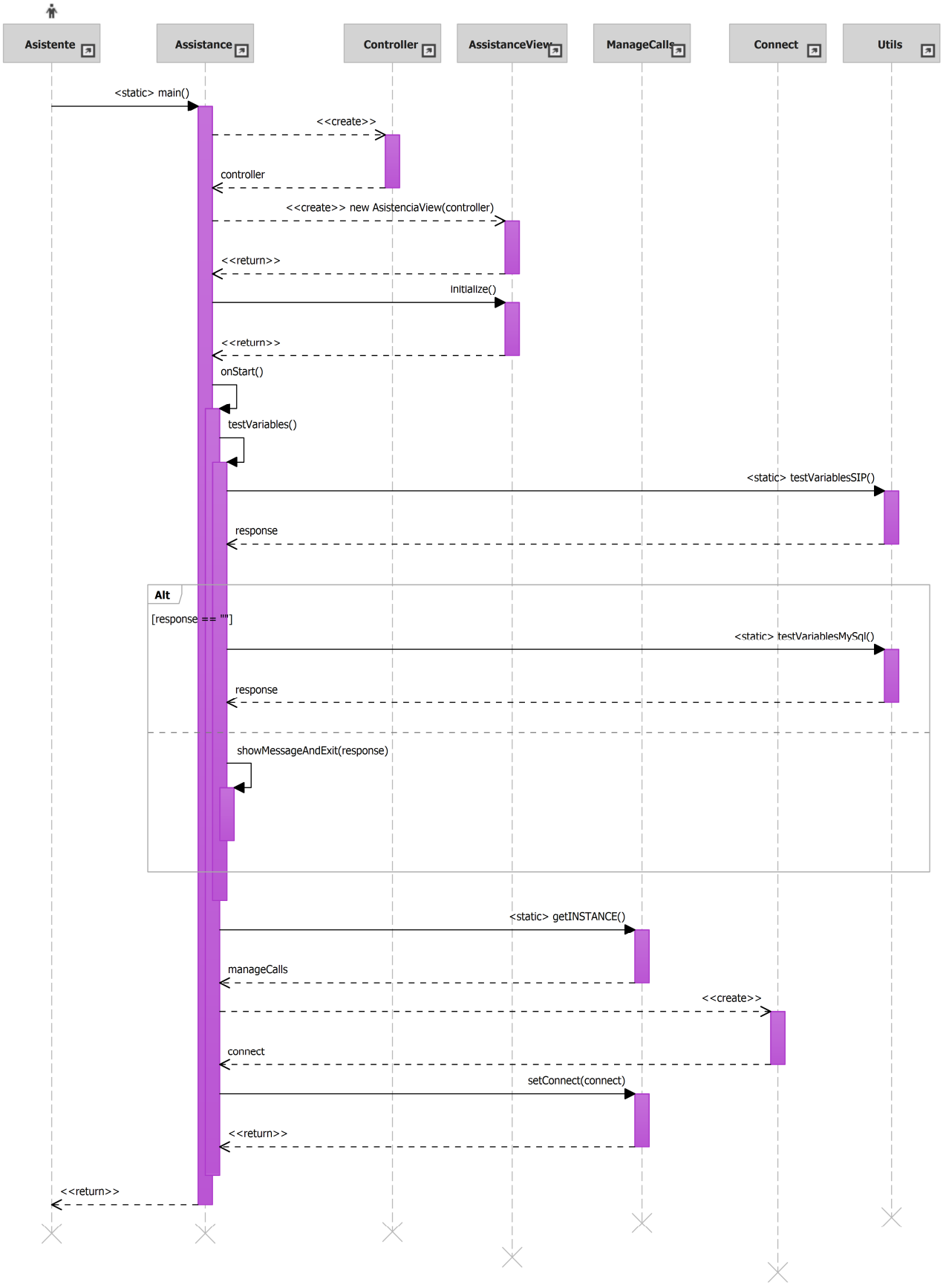


Ilustración 12: sd Iniciar.

4.3.2 Parte kiosco

En este apartado presentaremos los diagramas más importantes y representativos de la parte Kiosco. Del mismo modo, en el anexo C podremos observar el resto de diagramas creados.

A lo largo del apartado podremos observar la presencia de diagramas compuestos, del mismo modo que anteriormente, los diagramas auxiliares también estarán en el anexo C.

4.3.2.1 Diagramas principales

- 01- Llamar. Identificador del diagrama: **sd Llamar**.

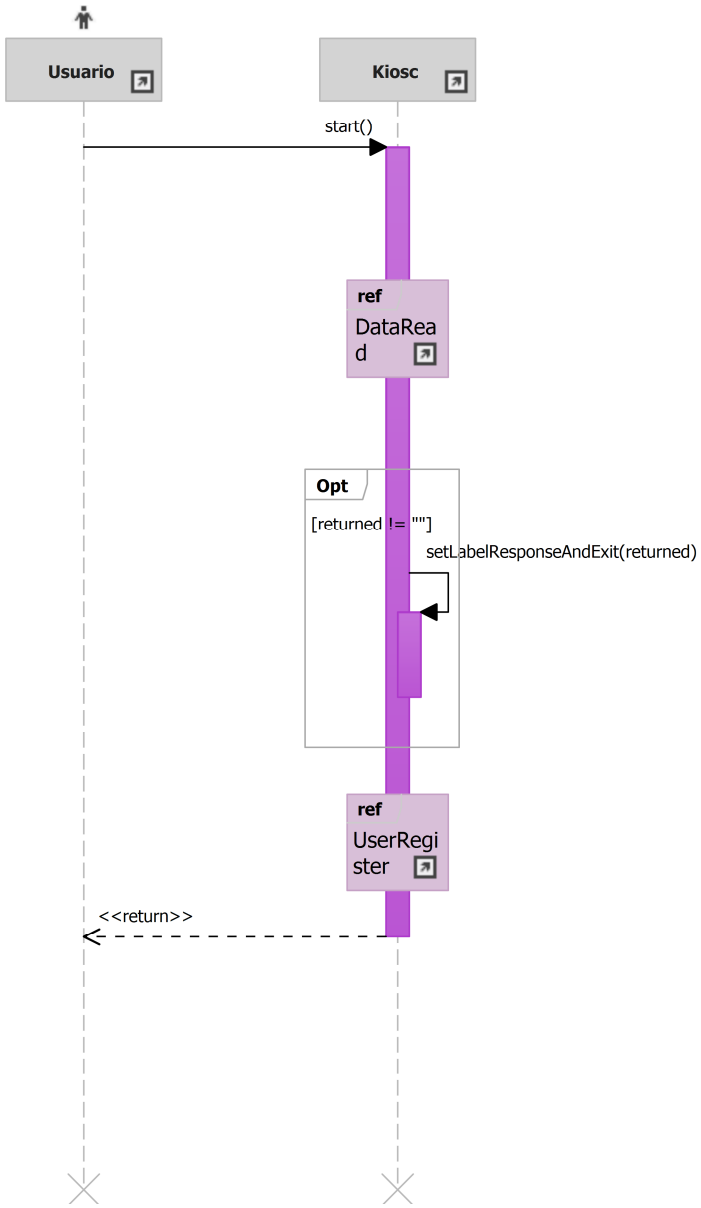


Ilustración 13: sd Llamar.

- 02- Registro usuario. Identificador del diagrama: **sd RegistroUsuario**.

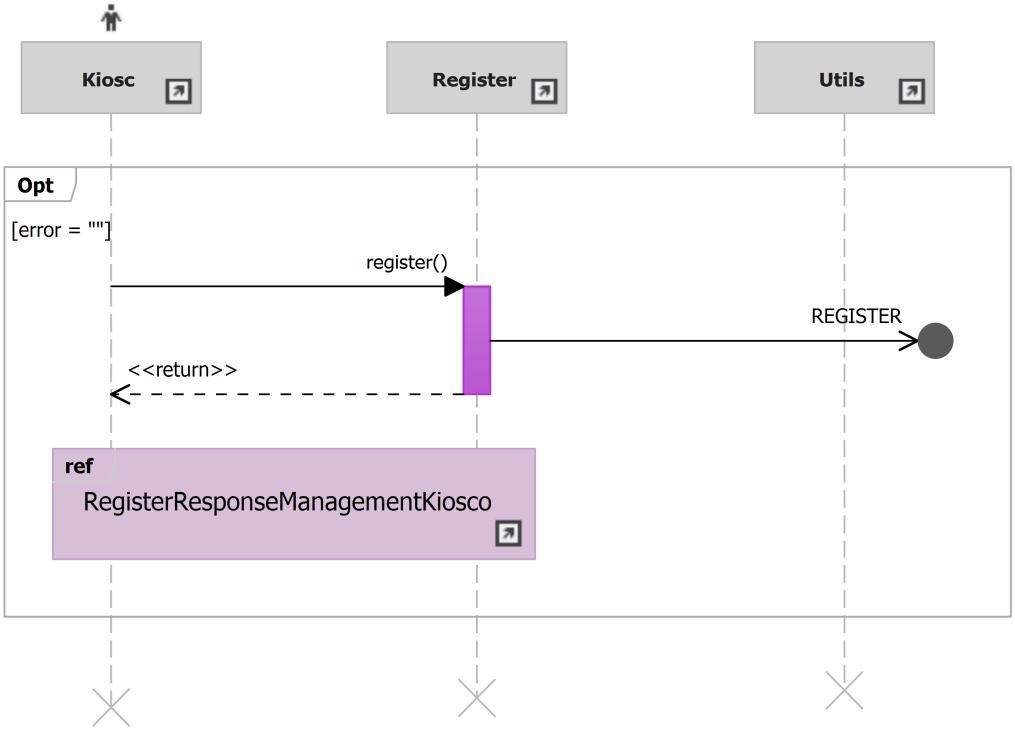


Ilustración 14: sd RegistroUsuario.

- 05- Cancelar llamada. Identificador del diagrama: **sd CancelarLlamada**.

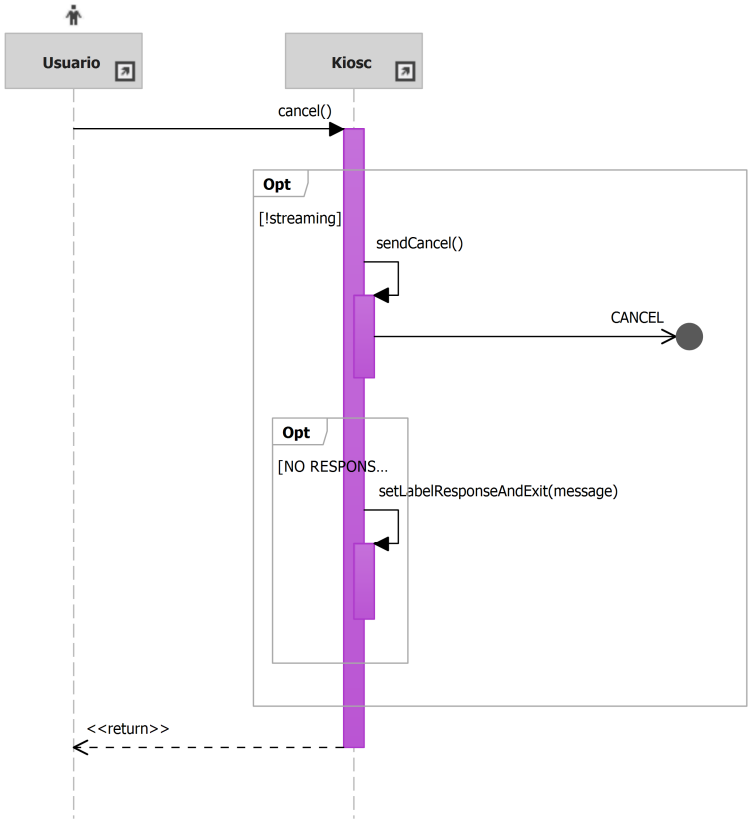


Ilustración 15: sd CancelarLlamada.

- 18- Empezar llamada. Identificador del diagrama: **sd EmpezarLlamada**

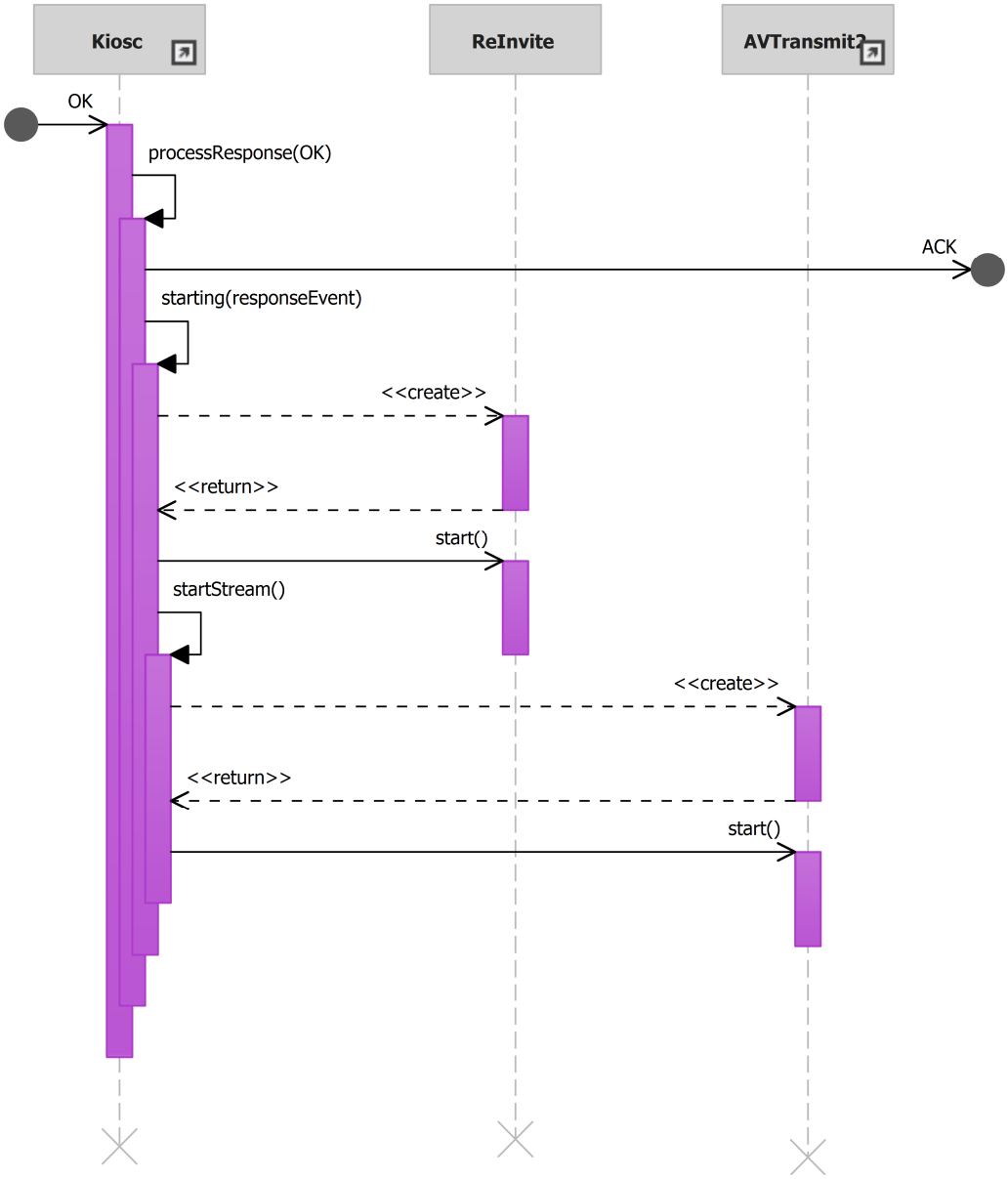


Ilustración 16: sd EmpezarLlamada.

- 27- Finalizar llamada asistente. Identificador del diagrama: **sd FinalizarLlamadaAsistente**.

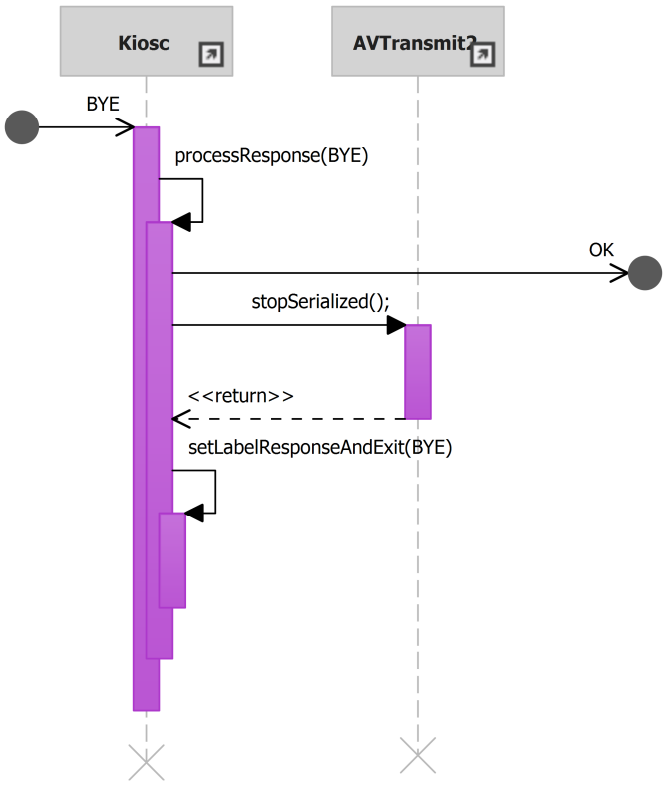


Ilustración 17: sd FinalizarLlamadaAsistente.

4.4 Diagramas de estados

Los diagramas de estados representan los cambios que se pueden realizar sobre los objetos que componen un sistema, del mismo modo también se puede observar las acciones que provocan estos cambios. Dicho de otro modo, un diagrama de estados describe el comportamiento de un objeto como un número de transiciones entre estados (Jacobson, Christerson, Jonsson, & Gunnar, 1992).

A lo largo de este punto presentaremos el diagrama de estado relacionado con la clase más importante de nuestro sistema y sobre la cual se pueden definir con gran claridad los posibles estados de una instancia de *Call* y las acciones que tienen efecto sobre ellas. Debemos diferenciar entre el estado de una llamada en el entorno de ejecución y el estado almacenado en la base de datos respecto una llamada.

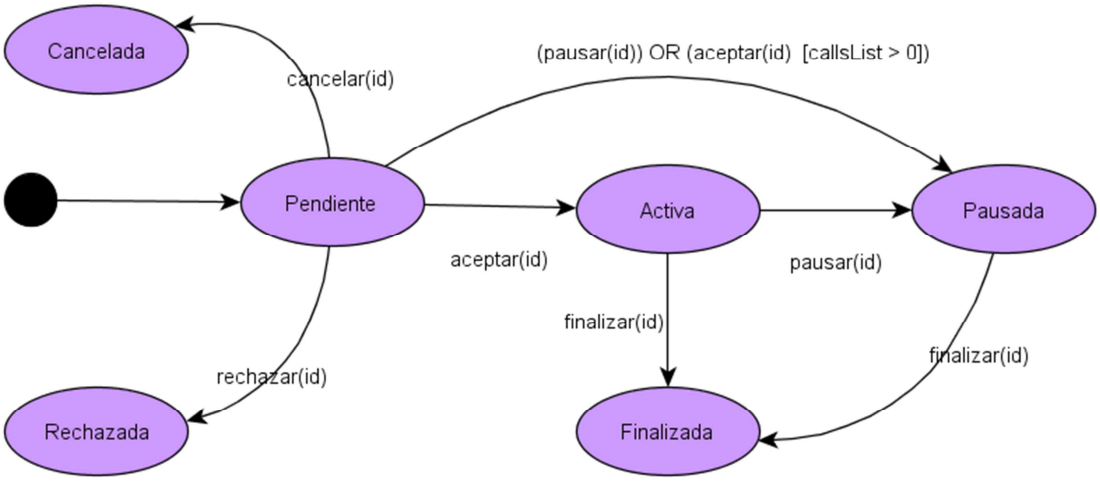


Ilustración 18: Diagrama de estados para la clase *Call*.

En la Ilustración 18, claramente observamos que se dispone de 6 estados: **Pendiente, Activa, Pausada, Finalizada, Rechazada** y **Cancelada**. Sobre una instancia de las acciones que modifican el estado de esta son: **cancelar, pausar, aceptar, rechazar** y **finalizar**. El escenario ideal de ejecución con el objetivo de ofrecer una atención al cliente de alto nivel seguiría el flujo de estados que muestra la Ilustración X.

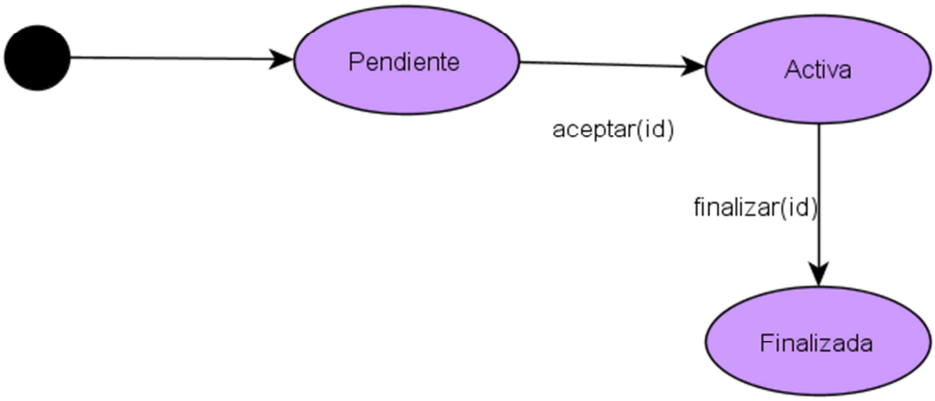


Ilustración 19: Diagrama de estados de la clase *Call* para un escenario ideal.

No obstante como hemos comentado en el párrafo anterior se trata del escenario ideal donde todos los factores que pueden afectar al trato del cliente sean positivos, este escenario es muy poco probable si se tiene un sistema con una gran cantidad de posibles usuarios con necesidad de atención. Un escenario más realista implica posibles cambios en los estados, de este modo podríamos decir que el conjunto de estados con mayor probabilidad es el que muestra la Ilustración 20.

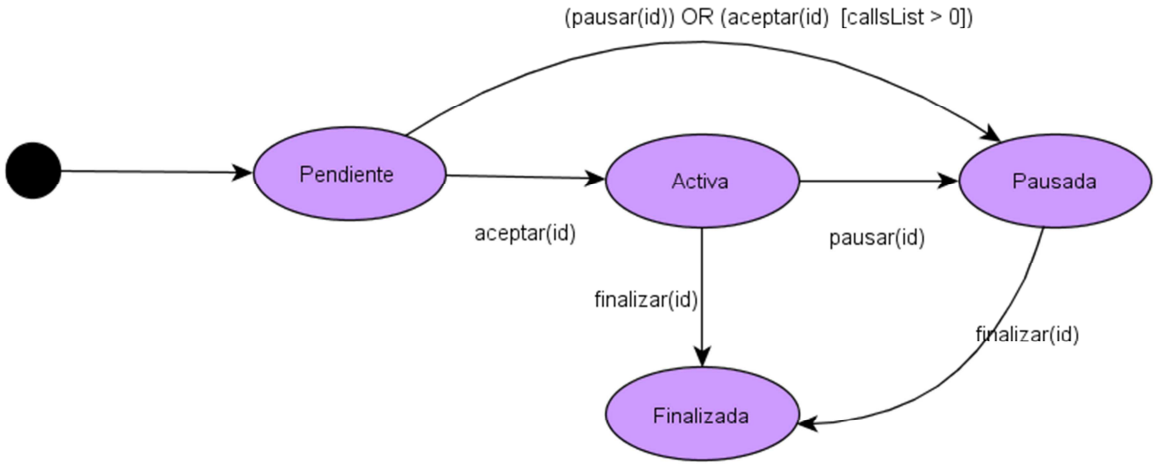


Ilustración 20: Diagrama de estados de la clase *Call* en un escenario realista.

Siguiendo con los estados posibles de una llamada debemos presentar el peor escenario posible, donde una llamada o bien es cancelada por el cliente o bien rechazada por el asistente por su incapacidad de atención en un tiempo dado (ver Ilustración 21).

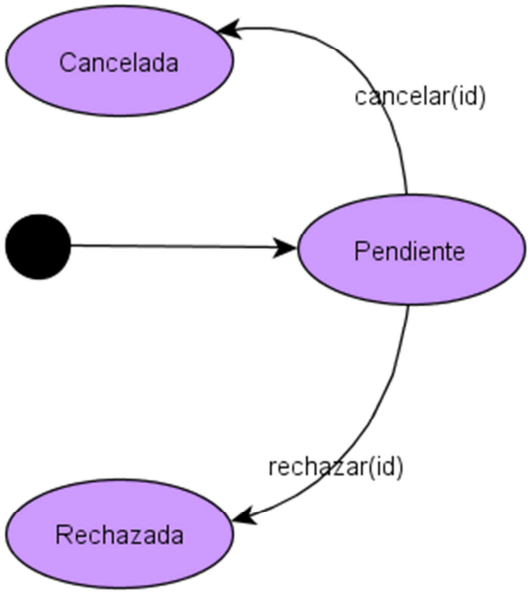


Ilustración 21: Diagrama de estados de la clase *Call* en el peor escenario.

5. Tecnología

Para el desarrollo de nuestra aplicación hemos utilizado distintos tipos de tecnologías de diversa naturaleza con el fin de ofrecer y posibilitar una mejor forma de trabajo. De este modo, aparte de las tecnologías propiamente necesarias para el funcionamiento de nuestra aplicación como son JAVA, el protocolo SIP y el SDP, la librería LibJitsi y el servidor Asterisk también hemos utilizado el entorno de desarrollo ampliamente conocido Eclipse y la herramienta de control de versiones Subversión mediante un plugin en el entorno.

5.1 JAVA

El lenguaje en el que nuestra aplicación ha sido desarrollada es JAVA. Se trata de un lenguaje de programación orientado a objetos con la mayoría de su sintaxis derivada de los lenguajes C y C++. Ha sido diseñado para ser multiplataforma y ser capaz de ofrecer navegabilidad por las redes de forma segura y una gran funcionalidad para remplazar código nativo de cada máquina (Niemeyer & Knudsen, 2013). Otro detalle importante de JAVA es la presencia del recolector de basura, este se encarga liberar la memoria de aquellos objetos que han sido creados pero que en cierto momento dejan de ser referenciados.

La independencia del sistema operativo se logra gracias la máquina virtual de JAVA (JVM), esta se instala en los equipos donde se quiera ejecutar software escrito en JAVA y ella se encarga de realizar las comunicaciones necesarias con el sistema operativo. Por otro lado para el desarrollo de software es necesario que el desarrollador tenga instalado el entorno de trabajo de JAVA (JDK).

5.2 SIP/SDP

El protocolo SIP fue creado por el *Internet Engineering Task Force* (IETF) se trata de un protocolo en la capa de aplicación para la creación, modificación y terminación de sesiones con uno o más participantes (Dennis Baron, 2015). Además permite la utilización del protocolo SDP para la definición del contenido multimedia de las sesiones. Por otro lado también se puede complementar con el protocolo RTP encargado de transportar el audio y el video dentro de una sesión SIP. El aspecto más importante a destacar es la capacidad de determinar la ubicación de los usuarios permitiendo así la comunicación con estos. En la Ilustración 22 se puede observar las relaciones entre los protocolos descritas previamente (Dennis Baron, 2015):

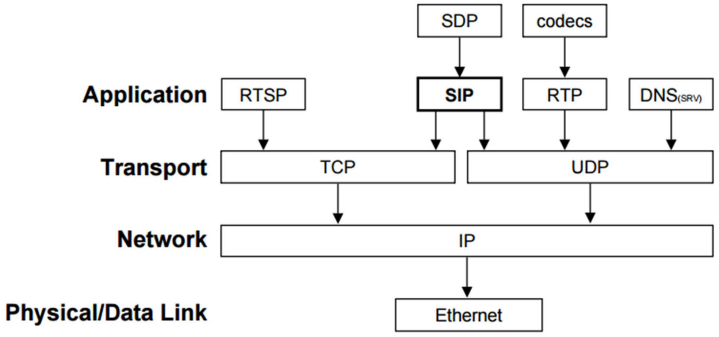


Ilustración 22: Esquema de protocolos por capas.

El protocolo SIP posee una estructura cliente-servidor y es transaccional, es decir, los usuarios realizan peticiones a los servidores y estos envían respuestas de vuelta. Por defecto el puerto utilizado para estas peticiones es el 5060, aunque este puede ser modificado. Debemos diferenciar que el protocolo SIP se encarga de establecer la comunicación entre puntos mientras que el *Session Description Protocol* (SDP) tiene la responsabilidad de definir todos aquellos parámetros multimedia que se deben tener en cuenta para la comunicación multimedia.

Dentro del protocolo SIP se distingue entre dos tipos de usuarios *User Agent Clients* (UAC) y *User Agent Servers* (UAS) el primero es aquel usuario que realiza una petición el segundo es el que la recibe. La forma de identificar a cada usuario es mediante la *Uniform Resource* Identifier (URI), que posee la siguiente estructura: sip:bob@biloxi.com, donde biloxi.com es el dominio del servidor SIP para el usuario Bob.

Además podemos distinguir entre dos tipos de servidores SIP: proxy y de redirección. El servidor proxy forma parte del camino que siguen los mensajes entre el UAC y el UAS, por el contrario el servidor de redirección simplemente indica el camino a seguir de los mensajes (solo interviene una vez).

5.2.1 Mensajes SIP/SDP

Una vez explicados los aspectos básicos del protocolo SIP vamos a realizar una breve explicación de la estructura de los mensajes, tanto peticiones como respuestas, que intervienen para establecer una sesión típica. También analizaremos los flujos de mensajes SIP típicos para acciones como el registro, la petición para establecer una sesión SIP, la cancelación y el rechazo de esta

5.2.1.1 Estructuras de los mensajes

Como hemos comentado anteriormente el protocolo SIP dispone de dos tipos de mensajes: peticiones (*requests*) y respuestas (*responses*). Ambos tipos disponen de la misma estructura: una línea de comienzo, una o más cabeceras, una línea en blanco indicando el fin de las cabeceras y un cuerpo del mensaje opcional.

5.2.1.1.1 Peticiones

Las peticiones SIP se distinguen por tener una línea de petición en el sitio de la línea de comienzo, dicha línea contiene el nombre del método, una Request-URI y la versión del protocolo separada por un espacio (SP). La línea de petición termina con un retorno de carro y un salto de línea, conocido como CRLF. A continuación vemos una tabla con la información relativa a los campos anteriores, no obstante en el apartado 6 explicaremos en más detalle los métodos utilizados.

Request-Line = Method SP Request-URI SP SIP-Version CRLF

Campo	Información
Método	La especificación del protocolo define seis métodos: *REGISTER*, *INVITE*, *ACK*, *BYE*, *CANCEL* y *OPTIONS*.
Request-URI	Indica el usuario o servicio a quien va dirigida la petición.
Versión SIP	Versión del protocolo utilizada.

Tabla 2: Estructura de las peticiones SIP.

5.2.1.1.2 Respuestas

Las respuestas SIP se distinguen por tener una línea de estado como su línea de comienzo. Una línea de estado está formada por la versión del protocolo SIP que se utiliza, un estado-número y su texto asociado, todos estos elementos separados por un espacio. La línea de estado finaliza con un CRLF. A continuación podemos observar una tabla con información acerca de los estados admitidos en el protocolo:

Status-Line = SIP-Version SP Status-Code SP Reason-Phrase CRLF

Estado	Información
1XX	Respuesta provisional.
2XX	La acción ha sido recibida, entendida y aceptada.
3XX	Redirección.
4XX	Error en el cliente. Errores de sintaxis o bien no puede ser procesada por el servidor.
5XX	Error en el servidor. El servidor no puede procesar una petición aparentemente correcta.
6XX	Error global. La petición no puede ser procesada por ningún servidor.

Tabla 3: Estados de las respuestas SIP

5.2.1.1.3 Cabeceras

Los campos de cabecera siguen el formato genérico que aparece en la sección 2.2 del RFC 2822. Cada cabecera consiste en el nombre de esta seguida de ":", un espacio en blanco y el valor de esta, en caso que queramos añadir algún parámetro más a una cabecera usaremos el carácter ";" seguido del nombre del parámetro el símbolo "=" y el valor de este último.

*field-name: field-value *(;parameter-name=parameter-value)*

5.2.1.1.3.1 Peticiones

Una petición SIP válida creada por un UAC debe contener como mínimo las cabeceras: *To*, *From*, *CSeq*, *Call-ID*, *Mx-Forwards* y *Via*. En la siguiente tabla se observan las cabeceras necesarias y otras con relativa importancia para el desarrollo de nuestra aplicación.

Campo	Información
Request-URI (línea de petición)	Contiene el identificador del usuario destino de la petición (a excepción de la petición *REGISTER*). Debe tener el mismo valor que la cabecera *To*.
To	Indica el destinatario de la solicitud, la dirección de registro (método *REGISTER*) o el recurso objetivo, mediante una o varias SIP URI.
From	Indica la identidad lógica del usuario iniciador de la petición.
Call-ID	Actúa como identificador único de los mensajes enviados. Debe tener el mismo valor para todos los mensajes enviados

	entre cada usuario.
CSeq	Sirve como forma de identificar y ordenar los mensajes recibidos. El método que aparece en esta cabecera debe ser el mismo que el de la petición.
Max-Forwards	Indica el número de saltos que un mensaje puede dar para llegar a su destino.
Via	Indica el transporte usado para la transacción e indica la localización donde se enviarán las respuestas. Debe contener el parámetro *branch* y se utiliza para identificar la transacción creada por la petición. Su función es importante en aquellos casos donde las peticiones navegan por varios servidores SIP.
Contact	Proporciona una o varias SIP URIS que pueden ser usadas como destino de futuras peticiones.

Tabla 4: Cabeceras protocolo SIP.

5.2.1.1.3.2 Respuestas

Las respuestas generadas por los UAS deben contener la misma cabecera *From* recibida en la petición para la que se está creando la respuesta, de igual modo para las cabeceras *Call-ID* y *CSeq* y *Via*. En el caso que la cabecera *To* no disponga de una etiqueta debemos añadirla como un nuevo parámetro. Esta etiqueta será utilizada para todas las respuestas generadas para la petición origen.

5.2.1.1.4 Cuerpo del mensaje

Gracias al protocolo SIP se permite la iniciación, modificación y finalización de sesiones con contenido multimedia, no obstante el protocolo SDP es el encargado de establecer el negoció del tipo de contenido, el formato de este y todos aquellos aspectos relacionados con el transporte e interpretación de los datos multimedia (Handley & Jacobson, July 2006). SDP no se encarga del transporte de los datos, tan solo es un protocolo para describir las sesiones multimedia. En el documento RFC4566 podemos encontrar todo aquello relacionado con este protocolo.

Para nuestro caso concreto se utiliza SDP dentro del cuerpo del mensaje SIP en las peticiones *INVITE*. De este modo podemos distinguir entre dos grupos para estas peticiones: **cabeceras** y **contenido del mensaje**. Las cabeceras siguen las definiciones y estructuras mencionadas en el apartado 5.2.1.1.3.1, seguidamente expondremos los detalles referentes al contenido del mensaje y por tanto los detalles que se han considerado de mayor importancia (para nuestras necesidades) del *Session Description Protocol*.

En el protocolo SDP se conoce como *Session Description* a aquel formato definido con la suficiente información para el descubrimiento y participación en sesiones multimedia. La estructura de este viene definida por cuatro puntos obligatorios:

- Nombre de la sesión y propósito.
- Tiempo activo de la sesión.
- El contenido multimedia que forma la sesión.

- Información necesaria para la recepción i envío de este contenido (direcciones, puertos, formatos...).

Definidos los puntos más importantes a destacar de la estructura de una *Session Description* podemos pasar a definir de forma menos abstracta su estructura. El primer y más importante punto es conocer que una sesión está formada por líneas que siguen el patrón:

<type>=<value>

Esta estructura se verá con mayor claridad a lo largo de este punto. No obstante, debemos resaltar la composición de las sesiones, formadas por dos o más secciones independientes: ***sesión-level*** y ***media-level***. Se dispone de una única *sesión-level* donde se encuentra toda aquella información general no perteneciente a la información propia del contenido multimedia, consecuentemente todo aquello relacionado con el envío y recepción multimedia estará en su correspondiente *media-level*. Como veremos en el ejemplo de uso del SDP se precisa de dos *media-level* para la comunicación de audio y video. Seguidamente se puede observar una estructura simplificada de una *Session Description* donde en morado se observa aquello relativo al *sesión-level* y en azul aquello perteneciente al *media-level*¸ los campos marcados con "*" son opcionales (ver Ilustración 23).

```
Session description
v= (protocol version)
o= (originator and session identifier)
s= (session name)
c=* (connection information—not required if included in
all media)

Time description
t= (time the session is active)

Media description, if present
m= (media name and transport address)
a=* (zero or more media attribute lines)
```

Ilustración 23: Estructura básica del protocolo SDP.

Siguiendo la estructura <type>=<value> encontramos distintas líneas con diferentes significados y funcionalidades. Con el objetivo de ofrecer una visión global sobre la importancia de las distintas líneas aquí presentes, se ofrece una breve descripción de cada una.

- ***Protocol versión ("v=")***

La primera línea, "v=", debe identificar la versión del protocolo utilizado, actualmente la 0.

- ***Origin ("o=")***

En segundo lugar observamos "o=", campo que ofrece el origen de la sesión. A diferencia del campo anterior este está formado por diversos sub-campos:

o=<username> <sess-id> <sess-version> <nettype> <addrtype>
<unicast-address>

Sub-campo	Información
<username>	Es el usuario que está registrado en el host origen.
<sess-id>	Se utiliza para la identificación de la sesión, junto con *<username>, <sess-id>, <nettype>, <addrtype>* y *<unicast-address>*. Se recomienda usar el formato *timestamp* para asegurar la uniquicidad
<sess-version>	Indica si se ha efectuado algún cambio sobre la anterior comunicación de la sesión.
<nettype>	Informa acerca del tipo de red utilizad.
<addrtype>	Contiene el tipo de dirección utilizada.
<unicast-address>	No se utiliza en nuestro proyecto. Para más información consultar el RFC4566.

Tabla 5: Subcampos de "o=". Protocolo SDP.

- ***Session name ("s=")***

Describe el nombre de la sesión. Tan solo debe existir un campo "s=" por sesión.

- ***Connection data ("c=")***

Cada sesión debe contener un campo "c=" en cada campo multimedia o bien un campo multimedia para toda la sesión (como será en nuestro caso). La estructura de este campo es la siguiente:

```
c=<nettype> <addrtype> <connection-address>
```

Sub-campo	Información
<nettype>	Identifica el tipo de red. Normalmente "*IN*" que representa Internet.
<addrtype>	Identifica el tipo de dirección. Los casos posibles son "IP4" y "IP6", que representan las dos versiones del protocolo IP.
<connection-address>	Este campo es opcional, representa la dirección de conexión.

Tabla 6: Subcampos de "c=". Protocolo SDP.

- ***Timing ("t=")***

Se encarga de identificar el inicio i final de una sesión. Estructurado en dos campos: *<start-time>* y *<stop-time>*. En aquellos casos en que ambos casos sean zero, identifica que una sesión es considerada como permanente.

- ***Media descriprion ("m=")***

Este campo identifica el inicio de una sección multimedia, el final de esta viene marcado por el inicio de una nueva sesión. Está estructurado de la siguiente forma:

```
m=<media> <port> <proto> <fmt> ...
```

Sub-campo	Información
<media>	Indica el tipo de contenido. Actualmente se soporta "*audio*", "*video*", "*text*", "*application*" y "*message*".
<port>	Puerto donde el contenido multimedia debe ser enviado.
<proto>	Identifica el protocolo de transporte del contenido.
<fmt>	Aporta información sobre el tipo de contenido.

Tabla 7: Subcampos de "m=". Protocolo SDP.

- ***Attributes ("a=")***

El campo atributo es la principal forma de aumentar las capacidades del protocolo SDP. Este campo aunque aparece en la Ilustración 23 como un atributo multimedia también puede usarse como uno de sesión o bien de ambos. La estructura del campo es la siguiente:

```
a=<attribute>:<value>
```

El campo "a=" consta de dos sub-campos no obstante, puede no aparecer el sub-campo <*value*>; de este modo sería un atributo binario, actuando como una propiedad. Para poder utilizar los atributos debemos tener en cuenta la capacidad de cada una de las partes de una sesión de interpretar estos de una forma correcta.

5.2.2 Escenarios habituales

Seguidamente presentaremos los escenarios importantes para el correcto establecimiento de sesiones en base a las necesidades requeridas por nuestro AsistenteVoIP obviando el resto de escenarios permisibles por el protocolo SIP.

5.2.2.1 Registro en el servidor

El primer paso para poder establecer una sesión entre un UAC y UAS es el registro de ambos en el servidor SIP. Mediante esta acción se consigue que ambos estén localizables para el envío y recepción tato de peticiones como respuestas.

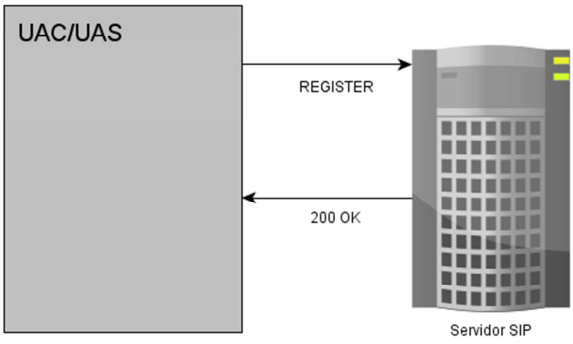


Ilustración 24: Registro sin contraseña

En la Ilustración 24 observamos la secuencia de peticiones y respuestas entre un UAC/UAS para el registro de este en un servidor SIP sin que este requiera la necesidad

de autorizarse, no obstante debido a motivos de seguridad y de control nuestro escenario seria el siguiente:

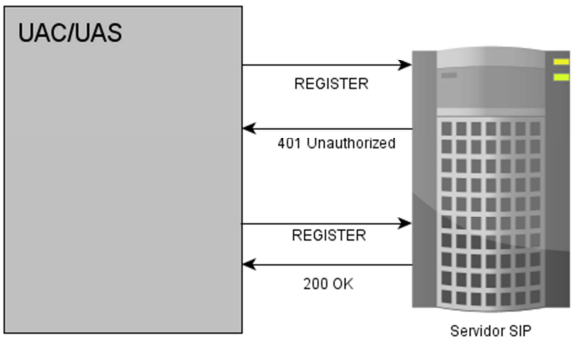


Ilustración 25: Registro con contraseña.

Exponiendo más detalle sobre la Ilustración 25 procedemos a explicar con más detalle el flujo de mensajes (in2eps, 2014):

1. El UAC/UAS envía una petición de registro a nuestro servidor SIP sin introducir ningún tipo de autenticación.
2. El servidor SIP (Asterix) manda una respuesta 401 *Unauthorized* con los datos necesarios para la autenticación del usuario. En la cabecera *WWW-Authenticate* encontramos los parámetros *nonce* y *realm* que nos permiten realizar el cifrado de la contraseña del usuario que quiere registrarse.
3. Con los datos ya cifrados el UAS/UAC crea una nueva petición de registro pero esta vez añade la cabecera *Authorization* que contiene la información cifrada.
4. El servidor responde 200 *OK*.

5.2.2.2 Petición de sesión

Con el objetivo de establecer una sesión entre dos usuarios, el UAC debe enviar una petición *INVITE* al servidor SIP indicando el usuario con el que quiere contactar. Para que este escenario funcione de la forma esperada tanto el UAC como el UAS deben haber realizado el registro SIP previamente.

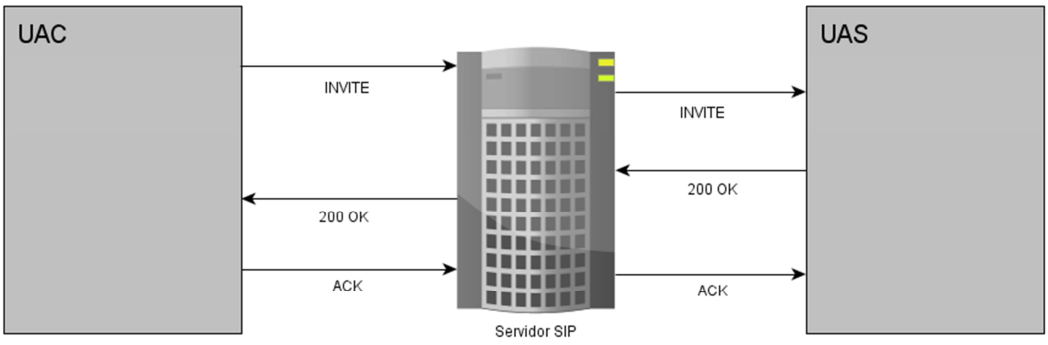


Ilustración 26: Petición de sesión.

Siguiendo la dinámica del apartado 5.2.2.1 procederemos a explicar con más detalle el flujo de mensajes (ver Ilustración 26) (in2eps, 2014):

1. El UAC envía una petición *INVITE* al servidor SIP indicando el usuario con el que se quiere establecer la sesión, y añadiendo al **contenido del mensaje** su información de *streaming* siguiendo el protocolo **SDP**. En este punto debemos tener en cuenta la configuración de nuestro servidor SIP, para nuestro caso un servidor Asterisk, ya que gracias a este podemos definir que no se acepten peticiones si no provienen de un UAC registrado.
2. Seguidamente el servidor SIP que en nuestro caso también funcionará como servidor RTP se encarga de **modificar todos aquellos datos referentes al protocolo SDP substituyéndolos por sus propios** datos, ya que este se utilizará como puente para el envío y recepción de audio y video.
3. El UAS acepta la petición de *INVITE* mediante un mensaje 200 *OK* donde el **contenido de este posee los datos de envío y recepción multimedia siguiendo el protocolo SDP**, del mismo modo que la petición *INVITE*. Como paso opcional, pero que aporta una gran riqueza a nuestra aplicación, el UAS puede generar una respuesta *RINGING*, previa a la aceptación *200 OK*, mediante la cual se comunica al UAC que la petición ha sido recibida en el otro extremo y se está concibiendo la opción de aceptación o denegación de la sesión.
4. Por último el UAC después de recibir el mensaje de aceptación responde con un ACK. Ésta es la única respuesta a una petición a la que se debe responder con un ACK.

5.2.2.3 Cancelación de sesión

El siguiente escenario representa aquellas situaciones en que un UAC decide finalizar/cancelar una petición de sesión antes de que esta sea aceptada (ver Ilustración 27).

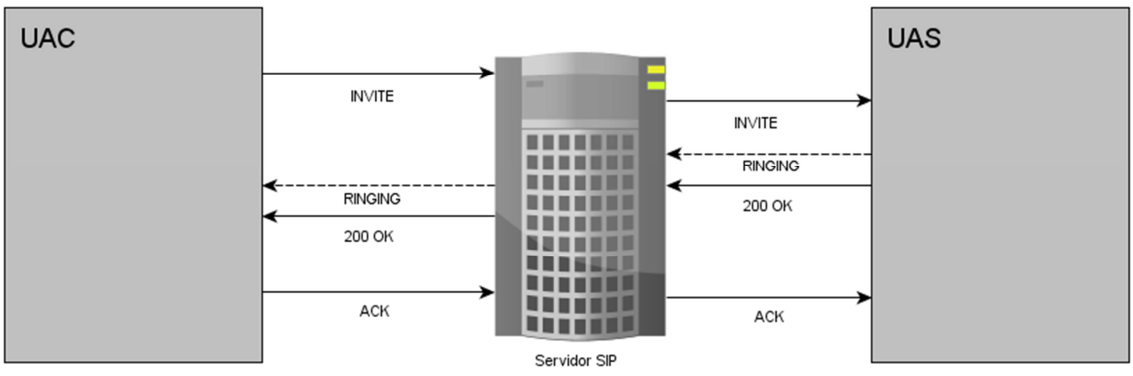


Ilustración 27: Cancelación de sesión.

Para finalizar en este punto 5.2.2.3 detallaremos el flujo de mensajes (in2eps, 2014):

1. Se realiza la acción 1 del escenario 5.2.2.2 Petición de sesión.
2. Antes de que el UAC reciba una respuesta del servidor SIP este envía una petición *CANCEL* con los datos de la sesión a finalizar.

Destacar que tanto la aceptación como el rechazo de la petición por parte del UAS no tienen efecto mientras el UAC no haya recibido la respuesta dada.

5.2.2.4 Rechazo de sesión

Otro caso donde se finaliza la sesión sin llegar a establecerse es en el caso en que el UAS decida rechazar la petición de *INVITE*, el UAS en vez de responder con un 200 OK responde con un *DECLINE* rechazando así el establecimiento de la sesión (ver Ilustración 28).

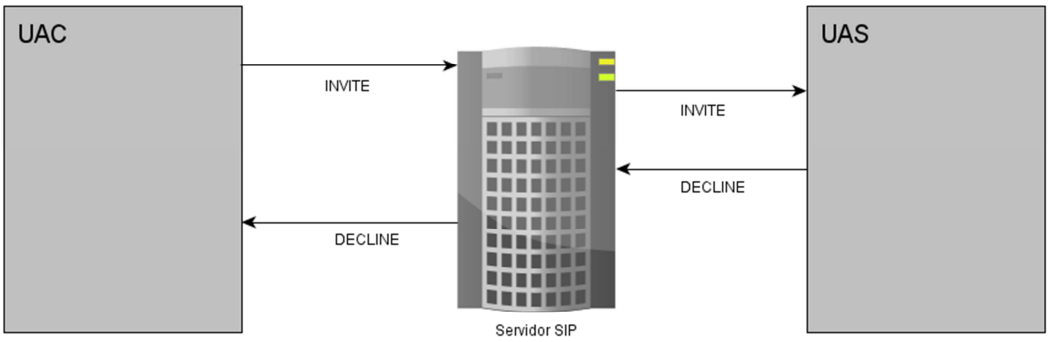


Ilustración 28: Rechazo de sesión.

El flujo de mensajes es el siguiente (in2eps, 2014):

1. Se realiza la acción 1 del escenario 5.2.2.2 Petición de sesión.
2. Cuando el UAS recibe la petición *INVITE* este responde con una respuesta *DECLINE* indicándose así que no se desea establecer comunicación.

5.3 LibJitsi

LibJitsi es una librería para JAVA con el objetivo de establecer comunicaciones seguras de audio y video. Permite a las aplicaciones la captura, reproducción, envío y recepción, codificación y decodificación. Originalmente formaba parte del programa Jitsi pero se decidió independizarla.

Tiene la capacidad de capturar videos tanto en Windows como en Mac OS X y Linux, puede codificar en H.264 y H.263, gran variedad de códigos de audio (Opus, SLIK, G.711 (PCMU, PCMA)...) y muchas más características que se puede consultar en: https://jitsi.org/Projects/LibJitsi.

Para el uso de esta librería tan solo es necesario descargarla de la página web e integrarla en nuestro entorno de desarrollo, además para su facilidad de uso también podemos encontrar una API con gran cantidad de información.

5.4 Asterisk

Como servidor SIP se decidió utilizar el servidor Asterisk debido a sus grandes funcionalidades, a su aparente facilidad de configuración y a su previo uso por parte de los técnicos de Tecatel S.L. Asterisk es un *framework* de código abierto para aplicaciones de comunicación, ofrece soluciones en tiempo real para el envío tanto de

audio como de video y además soporta los protocolos SIP y SDP que hemos utilizado para el establecimiento de sesiones multimedia.

Asterisk fue instalado en una máquina con el sistema operativo Ubuntu 12.04, para esta instalación se obtuvieron los ficheros necesarios de la página web y se siguieron los pasos allí definidos: https://wiki.asterisk.org/wiki/display/AST/Installing+Asterisk+From+Source.

Con la finalidad de configurar un servidor Asterisk que trabaje con el protocolo SIP los archivos principales que se deben modificar son: **sip.conf** y **extensions.conf**. De forma opcional también se puede modificar el archivo rtp.conf para establecer parámetros relativos al protocolo *Real-time Transport Protocol*(RTP) involucrado en el envío y recepción de audio y video.

5.4.1 sip.conf

El fichero sip.conf creado lo podemos encontrar en /etc/asterisk/sip.conf. En este archivo se deben definir los usuarios SIP que el servidor debe reconocer, además también se pueden definir los parámetros de configuración general para adecuar las necesidades de nuestra aplicación a las posibilidades permisibles del conjunto formado por Asterisk y SIP.

Tanto para añadir usuarios como para modificar los parámetros de estos o bien los parámetros generales del servidor Asterisk en lo referente al protocolo SIP debemos mantener la siguiente estructura:

[general]

parámetros de configuración general

[usuario1]
parámetros de configuración para el usuario

.

.

[usuarioN]

Parámetro	Información	Valores	General/Usuario
udpbindaddr/tcpbindaddr	Indican la dirección IP y el protocolo utilizado para la escucha de peticiones y respuestas SIP. También se puede definir el puerto.	0.0.0.0:5061 -> Escucha todas las IPs en el puerto 5061. 192.168.1.165 -> Escucha en la IP definida y el puerto por defecto 5060.	General
transport	Determina el protocolo de transporte	UDP TCP	General
videosupport	Indica al sistema	Yes	General/Usuario

	si este es capaz de enviar y recibir video.	no always	
directmedia	Indica al sistema si este debe permanecer entre los dos usuarios que quieren establecer una comunicación para el contenido multimedia.	Yes nonat update outgoing	General
nat	Indica cómo se debe actuar en aquellos casos que el cliente este detrás de una NAT.	No force_rport comedia auto_force_rport auto_comedia	General/Usuario
type	Indica el tipo de un usuario.	user friend peer	Usuario
secret	Contraseña del usuario.	-	Usuario
context	Indica el contexto que se debe seguir para las llamadas de cada usuario (los contextos se definen en extensions.conf).	**Nombre del contexto definido en extensions.conf**	Usuario
host	Indica la dirección IP del usuario.	dynamic -> asterisk no sabe la dirección del usuario hasta que este se registre. IP -> IP fija para un usuario.	Usuario

Tabla 8: Estructura sip.conf del servidor Asterisk.

En caso de querer obtener más información acerca de cada parámetro y de sus posibles valores se dispone de más información en el archivo ejemplo creado en la instalación.

5.4.2 extensions.conf

En este fichero se definen las acciones que debe realizar Asterisk cuando un usuario recibe una llamada. A pesar que este fichero ofrece unas grandes posibilidades en nuestro caso tan solo lo utilizaremos para enviar las peticiones de llamadas, que provienen de la parte del kiosco, a la parte asistente. Así pues, todo el peso del manejo de las llamadas será mantenido por la parte asistente. Por estos motivos no entraremos en mayor detalle en la configuración de las extensiones.

5.5 MySQL

MySQL ha sido el sistema de gestión de bases de datos seleccionado para la persistencia de toda aquella información relativa a las llamadas de los usuarios. Este sistema proporciona distintas cualidades que lo hacen destacar como son: la velocidad de trabajo, su facilidad de uso, el soporte del lenguaje ampliamente utilizado SQL (*Structured Query Language*), soporte multihilo, capacidad de trabajo de forma segura en la red, portabilidad, alta disponibilidad y por último de distribución y código abierto. Además se trata de un sistema ya utilizado por Tecatel y que dispone de una gran comunidad de desarrollo y consulta.

5.6 Subversion

Hemos utilizado un servidor de versiones Subversion (SVN) para controlar la evolución del software. Además se han creado ramas para separar cada versión operativa del software y así poder controlar mejor los cambios y asegurar siempre tener una versión con las mínimas funcionalidades. Los principales motivos por los que se ha seleccionado Subversion son su confiabilidad y a su gran facilidad de uso, además de que ya habíamos trabajado antes con él.

6. Desarrollo

En este capítulo presentaremos la información más relevante en el proceso de desarrollo, así como el diseño de la interfaz, fragmentos de código que sean de importancia para el correcto funcionamiento de nuestro AsistenteVoIP y algunas de las pruebas realizadas.

6.1 Proceso de desarrollo

Para el desarrollo utilizamos el proceso creado por la empresa Rational Software conocido como *Rational Unified Process* (RUP). RUP utiliza UML como lenguaje de modelado, aunque puede parecer una decisión trivial se trata de un aspecto importante. Gracias a UML se facilita el manejo de los requisitos, la modularización de los componentes, la visualización y entendimiento del software, control de cambios... Además utiliza los casos de uso para dirigir el proceso, es decir, estos se utilizan en cada etapa del ciclo de vida con objetivos distintos:

Etapa	
Especificación de requisitos	Capturar, clarificar y validar los casos de uso.
Análisis	
Diseño	Realizar los casos de uso.
Implementación	
Pruebas	Verificar si se satisfacen los casos de uso.

Tabla 9: Objetivos de los casos de uso por etapas.

RUP está basado en un desarrollo evolutivo e incremental ofreciendo en cada iteración un producto con unas características previamente definidas que serán mejoradas/modificadas en la próxima iteración. En nuestro proyecto se realizaron cuatro iteraciones siendo la última iteración la base de este proyecto.

La **primera iteración** tenía como objetivo el registro de los usuarios SIP en un servidor, con esta se buscaba asegurarse de la posibilidad de utilizar el protocolo SIP y confirmar el correcto funcionamiento de las librerías, además gracias al proceso RUP podemos identificar el origen del caso de uso "08- Registro asistente" en esta iteración. La **segunda** iteración buscaba comunicar visual y auditivamente dos máquinas distintas a través de la red, así confirmar el correcto funcionamiento de la librería LibJitsi. Con la **tercera** se buscaba establecer una sesión SIP/SDP entre la parte kiosco y la parte asistente. Y con la **cuarta** y última se buscaba unificar todos las tres anteriores con tal de ofrecer un sistema con las funcionalidades principales y añadir otras más complementarias.

6.2 Entorno de desarrollo

Para el desarrollo de la aplicación se ha utilizado un ordenador con el sistema operativo Windows 7 Home Premium de 64 bits con un procesador Intel® Core™ i7-3610QM CPU 2.30GHz y una memoria RAM de 8GB. Como dispositivo de video el sistema dispone de la cámara incorporada ASUS USB2.0 WebCam además también dispone de una tarjeta de sonido *Realtek High Definition Audio* con micrófono incorporado.

El entorno de programación utilizado ha sido Eclipse Indigo *Service Release* 2 y como hemos comentado anteriormente se ha utilizado el lenguaje de programación JAVA. Para consultar la configuración del entorno de desarrollo consulte el anexo 1.

6.3 Interfaz gráfica

En este apartado presentaremos la interfaz gráfica diseñada para cada una de las partes. "Un sistema software debe alcanzar su potencial máximo, es fundamental que su interfaz de usuario sea diseñada para ajustarse a las habilidades, experiencia y expectativas del usuario de sus usuarios previstos" (Sommerville, 2005) por este motivo ambas interfaces generadas están pensadas para sus usuarios correspondientes y de forma que sus interacciones sean muy intuitivos.

6.3.1 Parte asistente

La parte asistente posee una interfaz gráfica más completa que la de la parte kiosco debido al mayor número de acciones que un asistente puede realizar. Existen tres interfaces a destacar: la principal (ver Ilustración 29), la de notificación y la de comunicación. En la primera se identifican todas las llamadas recibidas, los dispositivos multimedia seleccionados y el estado del registro en el servidor.

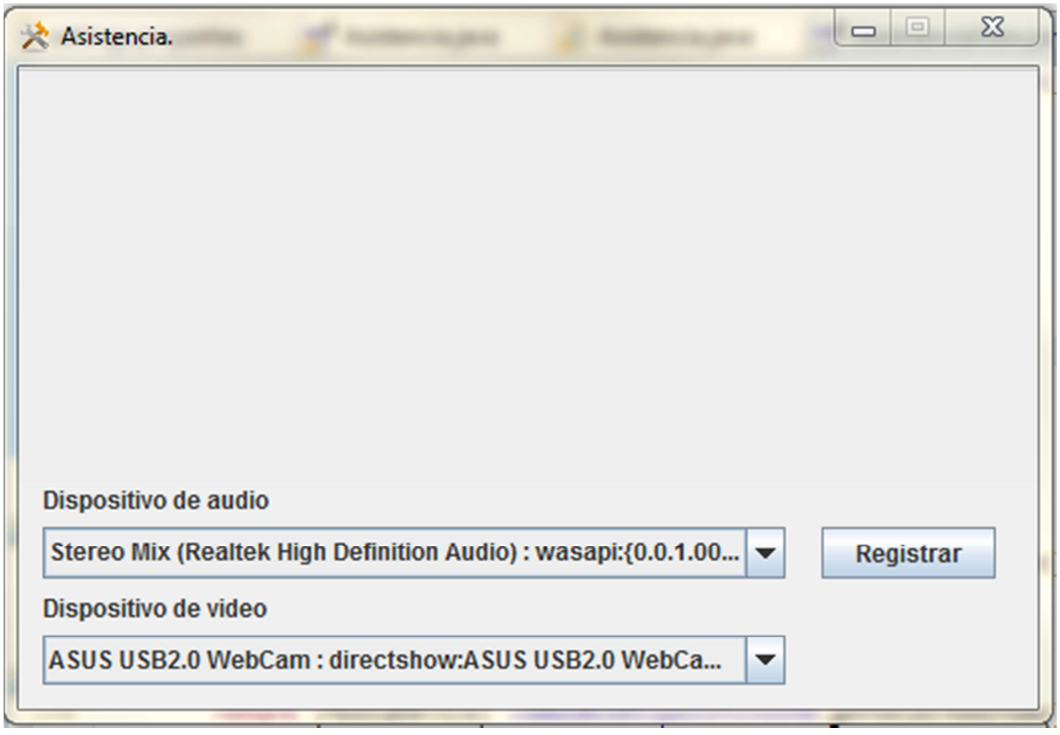


Ilustración 29: Parte asistente; interfaz principal.

La segunda se encarga de notificar la recepción de las llamadas recibidas, mostrando como información el hospital y el kiosco desde donde se llama. Además, presenta varios botones para aceptar una llamada, declinarla o bien ponerla a la espera (ver Ilustración 30).

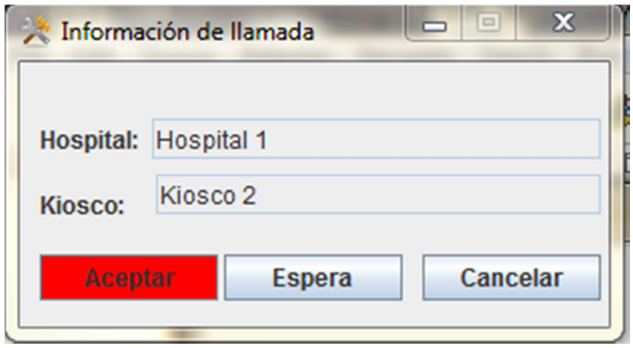


Ilustración 30: Parte asistente: interfaz de notificación.

Por último tenemos la interfaz de comunicación donde se puede observar al cliente de la parte kiosco y en menor medida en una esquina las imágenes capturadas por la cámara local. Esta interfaz ofrece una serie de posibilidades al Asistente: pausar la llamada, reactivar la llamada, visualizar escritorio, visualizar la cámara remota, refrescar las imágenes obtenidas y finalizar llamada (ver Ilustración 31).

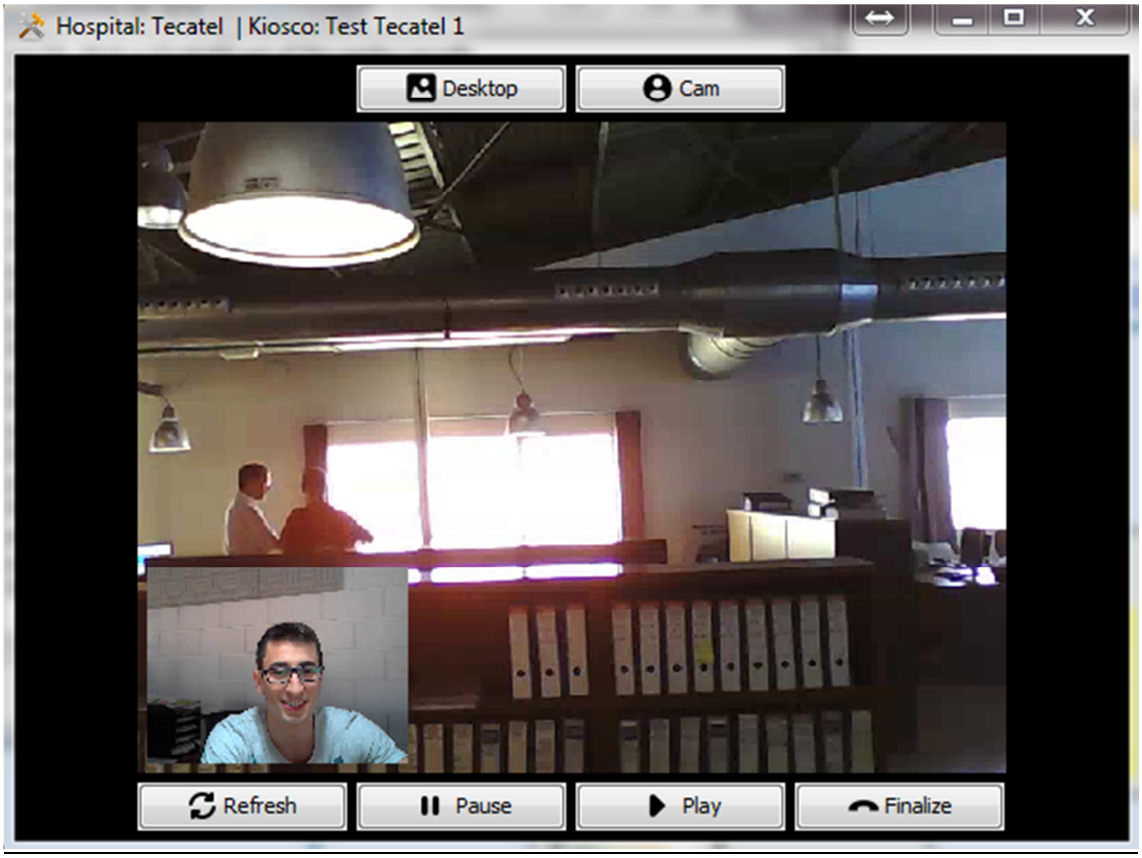


Ilustración 31: Parte asistente; interfaz de comunicación.

En definitiva se trata de un interfaz sencilla, en la que un usuario de un Kiosco no tiene mayor responsabilidad que la de pulsar un botón para iniciar una comunicación y con la misma acción finalizar ésta. Por otro lado el Asistente posee más funcionalidades y aunque este usuario del sistema no sea inexperto, se ha realizado un interfaz dinámica e intuitiva.

6.3.2 Parte Kiosco

La parte kiosco tan solo tiene dos funcionalidades. La primera se encarga de establecer la comunicación visual del cliente y el asistente, y la segunda tan solo frece al usuario la finalización/cancelación de la llamada. Además podemos dividir esta interfaz en dos fases: con la llamada ya establecida o por establecer.

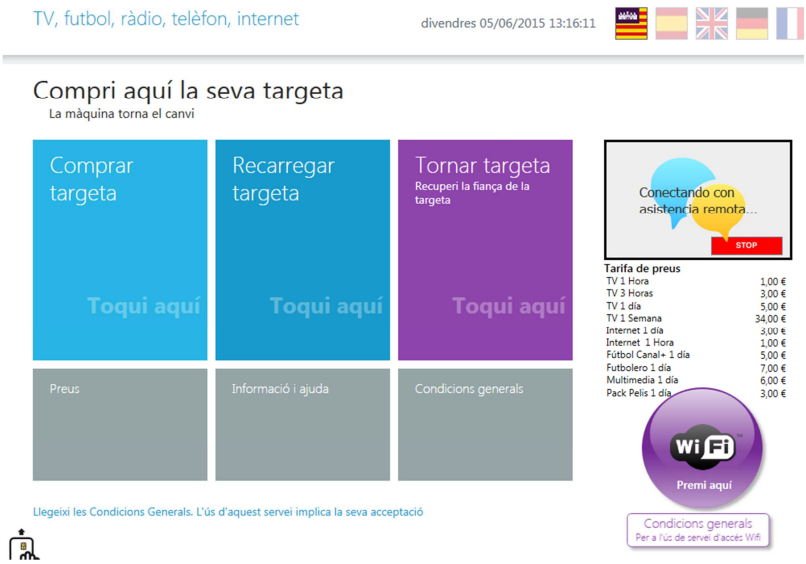


Ilustración 32: IGU parte cliente; llamada por establecer.

Con la llamada ya establecida se aprecian las imágenes recibidas desde la parte asistente y en una esquina, en menor tamaño, se observan las propias imágenes capturadas por la cámara del kiosco. Como se puede observar en ambas ilustraciones (ver Ilustración 32 e Ilustración 33) se trata de una interfaz sencilla e intuitiva que ofrece a los clientes las funcionalidades preestablecidas.

Ilustración 33: IGU parte cliente; llamada establecida.

6.4 Desarrollo de implementación

A lo largo de este apartado comentaremos aquellos aspectos más relevantes de nuestra aplicación pertenecientes a la etapa de desarrollo. De este modo analizaremos las dos partes de nuestra aplicación, explicando las clases más importantes de cada una y ofreciendo partes del código fuente que se crean de interés. También ofreceremos una descripción de aquellos ficheros relativos a la configuración de nuestra aplicación.

6.4.1 Parte asistente

Seguidamente podremos observar el código JAVA de aquellas clases principales de la parte asistente. De estas clases hemos seleccionado las porciones de código que se creen más importantes para el correcto funcionamiento del AsistenteVoIP. Dicho de otro modo, a continuación observaremos los puntos más importantes de la implementación de la parte con la que interactúa el asistente.

1. **Assistance.java**

Se trata de la clase principal de nuestra parte asistente. Tiene la responsabilidad de crear todos aquellos objetos necesarios para el establecimiento de sesiones SIP, además de ser el *SipListener* que hemos comentado en apartados anteriores. Otra tarea que le corresponde es la lectura de los datos almacenados en el fichero de configuración.

```
* Usuario con el que se registrará el asistente. El parametro de lectura del fichero es: “user”.
 */
public static final String USERNAME = Utils.readParameter(“user”);
/**
 * Servidor donde SIP. Parametro de lectura: “server”.
 */
public static final String SERVER = Utils.readParameter(“server”);
/**
 * Puerto local de envio y recepcion audio. Parametro de lectura: “localportaudio”.
 */
public static String LOCALPORTAUDIO = Utils.readParameter(“localportaudio”);

/**
 * Puerto local de envio y recepcion video. Parámetro de lectura: “localportvideo”.
 */
public static String LOCALPORTVIDEO = Utils.readParameter(“localportvideo”);

/**
 * Password del usuario para el registro SIP. Parámetro de lectura: “password”.
 */
public static final String SIPPASSWORD = Utils.readParameter(“password”);
/**
 * Puerto local de comunicación SIP. Parametro de lectura “localsipport”.
 */
public static final String LOCALSIPPORT = Utils.readParameter(“localsipport”);
/**
 * Puerto del servidor de comunicación SIP. Parámetro de lectura: “serversipport”.
 */
public static final String SERVERSIPPORT = Utils.readParameter(“serversipport”);
/**
 * Dirección SIP formado por el siguiente conjunto: “sip:”+<code>USERNAME</code>+”@”+<code>SERVER</code>.
 */
public static final String SIPADDRESS = “sip:” + USERNAME + “@” + SERVER;
/**
 * Indica si se guardan los datos de las llamadas en la base de datos.
 */
public static final String SAVECALLS = Utils.readParameter(“savecalls”);
```

Ilustración 34: Assistance; lectura de datos desde el fichero de configuración.

Como se observa en la imagen Ilustración 34 se leen aquellos parámetros del archivo de configuración relativos al usuario SIP dado de alta en el servidor, a la conexión con el servidor y a aquellos aspectos importantes a la hora del envío y recepción de audio.

En la Ilustración 35 se presenta el método encargado de inicializar los objetos necesarios para las comunicaciones SIP, no obstante la primera tarea que se realiza es la comprobación de la estructura de las variables, es decir, si una variable debe ser una IP debe cumplir su estructura, de igual forma para aquellas que deban tener un valor numérico. Destacar que estos objetos han sido creados como objetos de la clase *Assistance*, no obstante son públicos y de acceso estático para el posible uso de estos desde cualquier clase del proyecto.

```
//Testeo de las variables cargadas previamente. En caso de un error en alguna de las variables la
aplicación finaliza.
testVariables();//Si las variables son correctas se cargan los puertos de audio y video.
portVideo = Integer.parseInt(LOCALPORTVIDEO);
portAudio = Integer.parseInt(LOCALPORTAUDIO);
Assistance.ip = Utils.calcularIP();
// Creación de la fábrica SIP y definición de su ruta.
Assistance.sipFactory = SipFactory.getInstance();
Assistance.sipFactory.setPathName("gov.nist");
// Creacion de las propiedades de la pila SIP.
Assistance.properties = new Properties();
Assistance.properties.setProperty("javax.sip.STACK_NAME", "stack");
//Creación de la pila SIP. Y definición en las porpiedades del servidor SIP a utilizar.
Assistance.properties.setProperty("javax.sip.OUTBOUND_PROXY", SERVER+":"+SERVERSIPPORT);
Assistance.sipStack = Assistance.sipFactory.createSipStack(Assistance.properties);
// Creación de la fabrica de mensajes.
Assistance.messageFactory = Assistance.sipFactory.createMessageFactory();
// Creación de la fabrica de cabeceras.
Assistance.headerFactory = Assistance.sipFactory.createHeaderFactory();
// Creación de la fabrica de direcciones.
Assistance.addressFactory = Assistance.sipFactory.createAddressFactory();
// Creacion del punto de escucha SIP y bloqueo de la dirección IP, puerto y protocolo.
localSipPort = Integer.parseInt(LOCALSIPPORT);
Assistance.listeningPoint = Assistance.sipStack.createListeningPoint(Assistance.ip,
localSipPort, Assistance.protocol);

Assistance.sipProvider = Assistance.sipStack.createSipProvider(Assistance.listeningPoint);

// Añadimos nuestra clase como listener.
Assistance.sipProvider.addSipListener(this);
// Creacion de la direccion de contacto a utilizar en los mensajes SIP:
Assistance.contactAddress = Assistance.addressFactory.createAddress("sip:"
+ USERNAME + Assistance.ip);

Assistance.contactHeader = Assistance.headerFactory
        .createContactHeader(contactAddress);
```

Ilustración 35: *Assistance*; inicialización de los objetos necesarios SIP/SDP.

Como hemos comentado en el párrafo anterior Assistance.java se encarga de recibir y procesar las peticiones SIP que se reciban, no obstante si el asistente quiere recibir peticiones de llamada primero deberá registrarse en el servidor, de esto se encarga la clase *Register.java*. En la Ilustración 36 observamos el método que se encarga de procesar las peticiones recibidas y seguidamente el encargado de procesar las respuestas a peticiones enviadas.

```
  /**
   * Maneja las peticiones SIP recibidas.
   * @param requestEvent Evento que acompaña a la petición.
   */
  @Override
  public synchronized void processRequest(RequestEvent requestEvent) {
  Request request = requestEvent.getRequest();
  // Obtenemos el valor de la cabecera Call-ID, que nos sirve para identificar una llamada.
```

```
String callid = ((CallIdHeader)request.getHeader(CallIdHeader.NAME)).getCallId();
// Buscamos la llamada en nuestra clase de gestion de llamadas manageCalls.
int pos = manageCalls.searchCallByCallID(callid);
// Si la llamanda no se encuentra la añadimos y esta procesara la peticion.
// En caso de que la llamada ya este reconocida, esta misma procesara la peticion.
if(pos >= 0){
      manageCalls.getCall(pos).processRequest(requestEvent);
}else{
      Call call = new Call(requestEvent, this,
                  manageCalls, connect);
// Añadimos la peticion de llamada a nuestra gestión.
      manageCalls.addCall(call);
// La llamada procesa la petición.
      call.processRequest(requestEvent);
                  }
      }
/**
 * Procesa las respuesta recibidas. En caso que sea una respuesta relacionada
 * con el registro la clase Register se encargará de procesar la respuesta.
 * No obstante en el caso que sea una petición de tipo distinto se
 * le pasará la respuesta a la llamada a la que pertenece.
 * @param responseEvent Evento que acompaña a la respuesta.
 */
public void processResponse(ResponseEvent responseEvent) {
            // Se obitene la respuesta del evento.
Response response = responseEvent.getResponse();
CSeqHeader cseq = (CSeqHeader) response.getHeader(CSeqHeader.NAME);
// En caso de que sea una respuesta de autorización se responde
// con otra petición pero ahora con el campo necesario de autenticación.
if (response.getStatusCode() == Response.PROXY_AUTHENTICATION_REQUIRED
                  || response.getStatusCode() == Response.UNAUTHORIZED) {

       System.out.println("Received UNAUTHORIZED response.");
      register.sendAuthRequest(response);
      System.out.println("Sent REGISTER with auth header.");
// Si se trata de una respuesta a una petición de registro
// y además es un OK aceptándola se inicia el hilo para
// mantener activo el registro y se notifica al controlador.
} else if (cseq.getMethod().equals(Request.REGISTER)
                        && response.getStatusCode() == Response.OK) {

      System.out.println("Successfully REGISTRED.");
      register.setName("Register");
      register.start();
      controller.doRegistered();

// Si no se cumple ningún caso de los anteriores se busca
// la llamada origen de la petición para que
// procese la respuesta.
} else {
String callid = ((CallIdHeader) responseEvent.getResponse()
                              .getHeader(CallIdHeader.NAME)).getCallId();
int pos = manageCalls.searchCallByCallID(callid);
manageCalls.getCall(pos).processResponse(responseEvent);
            }
      }
```

Ilustración 36: *Assistance*; recepción de peticiones y respuestas.

2. Call.java

Junto con *Assistance* y *ManageCalls* es una de las clases más importantes del sistema, ya que representa la conexión de un asistente con un Kiosco determinado. Esta clase se encarga de mantener dinámicamente toda aquella información que se precisa para comunicarse con un Kiosco que ha establecido contacto previamente con la parte asistente. Tanto aquella información de comunicación SIP, como aquella relacionada con la comunicación SDP son mantenidas por la clase *Call*. También se encarga de manejar las peticiones que son recibidas y generar las respuestas adecuadas, siempre teniendo presente que es *Assistance* quien actúa como la "puerta" por donde se reciben

las peticiones. Y por último, dispone de la capacidad de controlar todos los estados posibles existentes procesando cada acción que decide realizar el asistente.

El primer punto a destacar es el método constructor. Este recibe como parámetros un evento perteneciente a la petición que ha generado la creación de la clase, una instancia de la clase *Assistance* para la comunicación de mensajes y por último una instancia de la clase *ManageCalls* encargada de la gestión de las llamadas.

```
/**
 * Constructor de la instancia de la clase Call. Esta clase se utiliza pera el manejo de las
 * llamadas recibidas.
 * @param requestEvent Evento de la petición de una llamada. Necesario para futuras
 * comunicaciones.
 * @param assistance Instancia de la classe principal Assistance. Ofrece la posibilidad de
 * acceder a los datos globales de la aplicación que no son estáticos.
 * @param manageCalls Instancia de la clase que se encarga del manejo de las llamadas.
 */
public Call(RequestEvent requestEvent, Assistance assistance
                ManageCalls manageCalls){
        this.assistance = assistance;
        this.requestEvent = requestEvent;
        this.request = requestEvent.getRequest();
        this.serverTransactionId = requestEvent.getServerTransaction();
        this.callId =
            ((CallIdHeader)requestEvent.getRequest().getHeader(CallIdHeader.NAME)).getCallId();
        this.manageCalls = manageCalls;
        this.manageMessage = new ManageMessages(this);
}
```

Ilustración 37: *Call*; constructor de la clase.

Como observamos en el fragmento de código de la Ilustración 37 se guarda la información necesaria de la petición originaria de una instancia de la clase *Call*, no obstante la instrucción a destacar es la que hace referencia al objeto *callId*, mediante esta se consigue obtener el identificador de cada llamada recibida para la futura gestión e identificación de las llamadas.

El segundo punto a destacar es el método encargado de enviar la aceptación de una petición de llamada. No obstante para poder llegar al punto en que una llamada pueda ser o no aceptada por el asistente debe haberse realizado el método *processInvite()* de forma exitosa. En la Ilustración 38 podemos observar un fragmento de este último método:

```
// Creación de la respuesta RINGING a partir de la petición recibida.
Response ringingResponse = Assistance.messageFactory.createResponse(Response.RINGING, request);
ContactHeader contactHeader = Assistance.headerFactory.createContactHeader(addressFrom);
response.addHeader(contactHeader);

// Añadimos una etiqueta para la cabecera ToHeader.
ToHeader toHeader = (ToHeader) ringingResponse.getHeader(ToHeader.NAME);
String toTag = new Integer((int) (Math.random() * 10000)).toString();
toHeader.setTag(toTag);

// Enviamos la respuesta al servidor.
serverTransactionId.sendResponse(ringingResponse);

[..]

this.fromTag = ((FromHeader)request.getHeader(FromHeader.NAME)).getTag();
this.toTag = toTag;
manageCalls.insertNewCall(this);
```

Ilustración 38: *Call*; fragmento de *processInvite()*.

Para la creación de la respuesta temporal *RINGING* utilizamos la petición recibida, de este modo la librería *jain-sip* genera una respuesta utilizando los datos recibidos. Seguidamente añadimos una etiqueta a la cabecera *ToHeader*, realizamos esta acción

debido a como se establece en la definición del protocolo SIP; cada sesión es identificada mediante la etiqueta de la cabecera *FromHeader* definida por el UAC, la etiqueta de la cabecera *ToHeader* y el contenido de la cabecera *Call-ID*. Por tanto, estos tres parámetros tienen una importancia especial de cara al establecimiento de sesiones a través del servidor Asterisk. Por último tan solo mencionar que una vez es procesada la petición de llamada por parte de la clase, se añade una instancia de esta a *manageCalls* que como hemos comentado se encarga de gestionar las llamadas.

3. ManageCalls.java

Para terminar con esta parte asistente presentamos la clase *ManageCalls*, encargada del manejo de las llamadas recibidas en la parte Asistente. Se encarga de mantener distintas llamadas, no obstante, tan solo una debe estar activada mientras el resto de ellas están pausadas, debido a que un asistente trabaja de forma secuencial y tan solo puede interactuar con un usuario del Kiosco.

La vertebral de *ManageCalls* reside en el método *processAction* que recibe como parámetros un entero identificando la acción que debe procesar la clase y un entero identificando la posición de una llamada en la lista de llamadas. Destacan dos variables dentro de *ManageCalls*: *actualIndexCall* y *callsList*, un entero y una lista de llamadas. El primer elemento se encarga de identificar dentro de la liste de llamadas a aquella que está activa a un momento dado del sistema. El segundo se encarga de almacenar instancias de la clase *Call*, representando el conjunto de llamadas disponibles en el sistema, independientemente del estado en que estas se encuentran.

```
/**
* Procesa una acción para una llamada determinada en un estado determinado. En conjunto de
* acciones posibles son:
* Acción == 2 --> Pone una llamada en espera / Aceptar una llamada manteniéndola a la espera.
* Acción == 4 --> Finaliza/Declina una llamada.
* Acción == 5 --> Acepta/Activa una llamada nueva/espera.
*
* Nótese que el sistema guarda el último estado detectado para evitar realizar múltiples veces
* la misma acción.
* @param action Acción a realizar.
* @param pos Posición de la llamada en la lista de llamadas.
*/
public synchronized void processAction(int action, int pos) {
        // TODO Auto-generated method stub
        Call call = getCall(pos);
                switch(action){
                        case 2: // Mantener llamada a la espera.
                                if(call.isFirstCall()){
                                        call.setNotificando(false);
                                        call.sendOutDialogMessage();
                                        try {
                                                Thread.sleep(500);
                                        } catch (InterruptedException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                        }
                                        call.startTimer();
                                        call.startStream(true);
                                        call.setStatus(0);
                                        call.setMessage(call.getMessage()+1);
                                        call.setFirstCall(false);
                                        call.setFirst(false);
                                }else{
                                        call.setNoActualCall();
                                        call.stopVideoAudio();
                                        call.sendSIPMessagePause();
```

```
                                        call.setStatus(0);
                                }
                                break;
                        case 4: // Colgar.
                                if(call.getDialog().getState() == DialogState.EARLY){
                                        call.setNotificando(false);
                                        call.decline();
                                        finalizeCall(call);
                                        doAction(3, call);
                                }else{
                                        Call aux = getActualCall();
                                        pos = indexOf(call);
                                        if(pos == actualIndexCall){call.setNoActualCall();}
                                        call.botoncancelar();
                                        call.restartOtherStream(aux);
                                        finalizeCall(call);
                                }
                                break;
                        case 5: // (re)Activar.
                                if(call.isFirst()){
                                        call.startTimer();
                                        call.setStatus(1);
                                        call.setNotificando(false);
                                        setActualCall(call);
                                        setActualIndexCall(pos);
                                        call.stopOtherStreams();
                                        try {
                                                Thread.sleep(500);
                                        } catch (InterruptedException e) {
                                                // TODO Auto-generated catch block
                                                e.printStackTrace();
                                        }
                                        doAction(1, call);
                                        call.setServed(true);
                                        call.setMessage(call.getMessage()+1);
                                        updateServed(call);
                                        call.setFirstCall(false);
                                        call.setFirst(false);
                                        call.startStream(false);
                                        break;
                                }else{
                                        call.stopOtherStreams();
                                        if(call.getMessage() >= 1){
                                                call.sendSIPMessagePlay();
                                        }
                                        call.setMessage(call.getMessage()+1);
                                        call.setStatus(1);
                                        doAction(1, call);
                                        call.reStartStream();
                                        if(!call.isServed()){
                                                updateServed(call);
                                        }
                                }
                                break;
                        }
                }
```

Ilustración 39: ManageCalls; método processAction().

Por tanto distinguimos entre tres acciones:

- **Acepta/Activa una llamada**: corresponde a la acción 5. Dentro de esta acción se distinguen dos subconjuntos:
    - **Aceptar**: al aceptar una llamada se cesa en la notificación audiovisual de esta, se responde la petición *INVITE* aceptándola, se actualizan los índices que indican la llamada activada y para finalizar se activa el *streaming* del contenido recibido para la llamada aceptada y se pausa el resto. Además se actualiza el estado de la llamada.

- **Activar**: en este caso, debido a que se está reactivando una llamada previamente pausada, tan solo se le comunica a la parte Kiosco la decisión y se reactiva el envío y recepción de audio y video, se actualizan los índices que marcan la llamada activada, se le notifica a la parte kiosco y se actualiza el estado de la llamada.

- **Pone una llamada en espera / Aceptar una llamada manteniéndola a la espera**: corresponde a la acción 2. Se distingue entre dos tipos de acciones, si la llamada ya ha sido aceptada y el caso contrario**:**
  - **No aceptada previamente**: sigue el mismo flujo que cuando se acepta una nueva llamada, no obstante, la diferencia radica en que ahora tan solo se inicia la recepción de video, y se le notifica a la parte kiosco que la llamada se va a mantener a la espera. No se actualizan los índices de las llamadas, ya que se mantiene activa aquella que previamente ya lo estaba.
  - **Aceptada previamente**: para este caso si se actualizan el índice que marca la llamada activada. A continuación se le notifica a la parte Kiosco la acción que debe realizar. Además, obviamente, se activa tanto el envío como la recepción de contenido multimedia. Y como en los casos anteriores se modifica el estado de la llamada.
- **Finaliza/Declina una llamada**: corresponde a la acción 4. Para este punto debemos distinguir entre si una llamada se ha aceptado antes de ser finalizada. A alto nivel no existe diferencia entre finalizar una llamada sin haber sido aceptada con aquella que ha sido aceptada. Pero a un nivel más bajo se distingue entre estos dos casos, al primero se le llama finalizar una llamada y al segundo declinar.
  - **Finalizar**: cuando una llamada es finalizada el primer paso consiste en el cese de envío de información multimedia, seguidamente el índice que indica cual es la llamada principal es vaciado y se le notifica a la parte Kiosco la decisión de finalizar la comunicación.
  - **Declinar**: cuando se declina una llamada simplemente se envía la respuesta *DECLINE* a la llamada y se actualiza su estado. Obviamente se cesa en la notificación audiovisual de esta.

6.4.2 Parte kiosco

Igual que para la parte Asistente a continuación presentaremos aquellas clases JAVA que se crea más importante para la parte Kiosco. No obstante debemos tener en mente que ambas partes forman un mismo sistema y por lo tanto existen porciones de código que se pueden observar en ambas partes pero adecuadas a su entorno.

1. Cliente.java

Forma la clase principal de la parte Kiosco. Igual que *Assistance.java* para la parte Asistente, se encarga de crear aquellos objetos necesarios para establecer comunicaciones utilizando el protocolo SIP. Además contiene aquellas variables de mayor importancia gracias a las cuales se pueden establecer la comunicación con el asistente o bien conectar con el servidor de base de datos para la lectura de ciertos parámetros.

```
/**
* Identificador del hospital.
*/
public static String HOSPITAL;// = Utils.readParameter(FILEINFO ,"hospital:");
/**
* Identificador del kiosco.
*/
public static String KIOSCO; // = Utils.readParameter(FILEINFO ,"kiosco:");
/**
* Code de audio a utilizar en la transmisión de audio.
*/
public static String AUDIOCODE;// = Utils.readParameter(FILESETTINGS ,"audiocode:");
/**
* Dispositivo de audio predeterminado.
*/
public static String AUDIODEVICE;// = Utils.readParameter(FILESETTINGS ,"audiodevice:");
/**
* Dispositivo de video predeterminado.
*/
public static String VIDEODEVICE;// = Utils.readParameter(FILESETTINGS ,"videodevice:");
/**
* Usuario SIP al que se realizará la llamada.
*/
public static String TOUSERNAME;// = Utils.readParameter(FILESETTINGS
,"touser:");//"asistenciatecatel";
/**
* IP del servidor SIP.
*/
public static String TOSERVER;// = Utils.readParameter(FILESETTINGS
,"toserver:");//"iptel.org";
/**
* Puerto de audio para el envio y recepcion de este.
*/
public static String LOCALPORTAUDIO;// = Utils.readParameter(FILESETTINGS
,"localportaudio:");//"5098";
/**
* Puerto de video para el envio y recepcion de este.
*/
public static String LOCALPORTVIDEO;// = Utils.readParameter(FILESETTINGS
,"localportvideo:");//"5100";
/**
* Usuario SIP desde el que se realizará la llamada.
*/
public static String FROMUSERNAME;// = Utils.readParameter(FILESETTINGS ,"fromuser:");
/**
* Password del usuario <code>FROMUSERNAME</code>.
*/
public static String SIPPASSWORD;// = Utils.readParameter(FILESETTINGS,"password:");
/**
* Puerto del servidor SIP.
*/
public static String SERVERSIPPORT;// = Utils.readParameter(FILESETTINGS, "serversipport:");
/**
* Puerto local SIP.
*/
public static String LOCALSIPPORT;// = Utils.readParameter(FILESETTINGS, "localsipport:");
```

Ilustración 40: *Cliente*: variables a destacar.

A diferencia de la clase *Asistente* en esta no se inicializan las variables debido a que estas pueden tener dos fuentes origen: la base de datos relacional o bien el archivo de configuración. Aunque no aparece en el fragmento presentado en la Ilustración 40, tras cargarse las variables, coincidiendo con el flujo de *Assistance*, se realiza la comprobación de su estructura para evitar errores en instantes futuros.

Siguiendo la estructura seguida en la clase *Assistance* destacan los métodos *processRequest* y *processResponse*. El primero se presenta en la porción de código de la Ilustración 41 y su función es la de procesar las petición que son recibidas.

```
/**
* Maneja la recepción de peticiones SIP. Al tratarse de la parte pasiva del sistema, las
* peticiones recibidas
* serán o bien BYE, CANCEL para finalizar la conexión, o MESSAGE para el manejo de las pausas
* de streaming.
* La petición UPDATE no se utiliza, a pesar que está reflejada en el código.
* @param requestEvent Evento de la petición.
*/
public void processRequest(RequestEvent requestEvent) {
        // Obtiene el objeto representante de la petición.
        Request request = requestEvent.getRequest();
        System.out.println("Got a request:"+request.getMethod());
        Response okResponse = null;

        if (request.getMethod().equals(Request.INVITE)) {
                //Si se reciben peticiones de llamada se ignoran.
        } else if (request.getMethod().equals(Request.BYE)) {
                recievedBye = true;
                ServerTransaction st = requestEvent.getServerTransaction();
                try {
                        if(st == null) st = sipProvider.getNewServerTransaction(request);
                        av2.stopSerialized(); // Finaliza la comunicación multimedia.
                        okResponse = messageFactory.createResponse(Response.OK, request);
                        st.sendResponse(okResponse); // Confirma la recepción de la petición.
                        System.out.println("Received BYE. \n Sent OK \n"+okResponse);
                } catch (SipException | InvalidArgumentException | ParseException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                setLabelResponseAndExit("<html>Asistentes ocupados,"+"<br>"+"intentelo más
                        tarde.<br></html>");
        } else if (request.getMethod().equals(Request.CANCEL)) {
                recievedBye = true;
                ServerTransaction st = requestEvent.getServerTransaction();
                try {
                        if(st == null) st = sipProvider.getNewServerTransaction(request);
                        okResponse = messageFactory.createResponse(Response.OK, request);
                        st.sendResponse(okResponse); // Confirma la recepción de la petición.
                        System.out.println("Received CANCEL. \n Sent OK \n"+okResponse);
                } catch (SipException | InvalidArgumentException | ParseException e) {
                        // TODO Auto-generated catch block
                        e.printStackTrace();
                }
                setLabelResponseAndExit("<html>Asistentes ocupados,"+"<br>"+"intentelo más
                        tarde.<br></html>");
        }else if(request.getMethod().equals(Request.MESSAGE)){
                processMessage(requestEvent, request);
        }
}
```

Ilustración 41: *Cliente*; método *processRequest()*.

Podemos diferenciar entre cuatro secciones dentro del método, cada una encarga de un tipo específico de petición:

- **Proceso de las peticiones *INVITE***: el sistema no está creado pensando que los kioscos puedan recibir llamadas, por tanto si se diera el caso en que en un Kiosco se recibiera una petición de este tipo, sería ignorada.

- **Proceso de las peticiones *BYE***: el sistema responde al servidor y finaliza toda comunicación con la parte asistente.
- **Proceso de las peticiones *CANCEL***: a diferencia del punto anterior el sistema notifica al usuario que no es posible atender su llamada y posteriormente finaliza.
- **Proceso de las peticiones *MESSAGE***: mediante este tipo de peticiones la parte asistente notifica al Kiosco de las acciones como la pausa y reanudación de una llamada, o bien, del dispositivo de video a visualizar.

El siguiente punto que destaca de esta clase es el método *processResponse*, este es complementario al método anterior y se encarga de procesar las respuestas recibidas.

```
/**
* Maneja las respuestas SIP recibidas. Se encarga de comunicar al cliente de si la
* comunicación podrá ser
* establecida, así como de controlar el registro del usuario SIP en el servidor.
*/
public void processResponse(ResponseEvent responseEvent) {
        // Obtiene el objeto representante de la respuesta.
        Response response = responseEvent.getResponse();
        System.out.println("Got a response \n"+response);
        ClientTransaction tid = responseEvent.getClientTransaction();
        CSeqHeader cseq = (CSeqHeader) response.getHeader(CSeqHeader.NAME);

        if (response.getStatusCode() == Response.OK) {
                if (cseq.getMethod().equals(Request.INVITE)) {
                        try {
                                dialog = responseEvent.getDialog();
                                System.out.println("response status  INVITE
                                        :"+response.getStatusCode());
                                Request ackRequest = dialog.createAck(cseq.getSeqNumber());

                                // Inicializa los elementos necesarios para el envio y
                                // recepción multimedia.
                                starting(responseEvent);
                                dialog.sendAck(ackRequest);
                                }
                        } catch (InvalidArgumentException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        } catch (SipException e) {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }else if (cseq.getMethod().equals(Request.REGISTER) &&
                                response.getStatusCode() == Response.OK) {
                        doRegistered();
                        register.start();
                        sendInvite();
                }
        } else if(response.getStatusCode() == Response.TEMPORARILY_UNAVAILABLE
                        || response.getStatusCode() == Response.REQUEST_TIMEOUT){
                System.out.println("Temporalmente no disponible.");
                setLabelResponseAndExit("Temporalmente no disponible.");
        }else if(response.getStatusCode() == Response.RINGING &&
                        cseq.getMethod().equals(Request.INVITE)){
                System.out.println("Received RINGING.");
        }else if(response.getStatusCode() == 183 &&
                        cseq.getMethod().equals(Request.INVITE)){
                System.out.println("Received 183: Session in progress!");
        }else if(response.getStatusCode() == 486 &&
                        cseq.getMethod().equals(Request.INVITE)){
                System.out.println("Received 486: BUSY!");
                setLabelResponseAndExit("<html>Asistentes ocupados,"+"<br>"+"intentelo más
                        tarde.<br></html>");
        }else if(response.getStatusCode() == Response.DECLINE &&
                        cseq.getMethod().equals(Request.INVITE)){
```

```
                System.out.println("Received DECLINE.");
                setLabelResponseAndExit("<html>Asistentes ocupados,"+"<br>"+"intentelo más
                        tarde.<br></html>");
        }else if (response.getStatusCode() == 503){
                btnCancelar.setEnabled(false);
                System.out.println("Received 503: Service Unavailable!");
                setLabelResponseAndExit("<html>Asistentes ocupados,"+"<br>"+"intentelo más
                        tarde.<br></html>");
        }else if (response.getStatusCode() == Response.PROXY_AUTHENTICATION_REQUIRED
                        || response.getStatusCode() == Response.UNAUTHORIZED) {
                if(firstAttempt){
                        try {
                                System.out.println("Received UNAUTHORIZED response.");
                                Request authrequest = register.registrarAuth(response);
                                inviteTid =
                                        sipProvider.getNewClientTransaction(authrequest);
                                inviteTid.sendRequest();
                                System.out.println("Sent REGISTER with auth header.");
                        } catch (TransactionUnavailableException ex) {
                                System.out.println(ex.getMessage());
                        } catch (SipException ex) {
                                System.out.println(ex.getMessage());
                        }
                }else{
                        System.out.println("Received 401 or 402: can not register");
                        setLabelResponseAndExit("<html>Asistentes ocupados,"
                                +"<br>"+"intentelo más tarde.<br></html>");
                }
        }
}
```

Ilustración 42: *Cliente*; método *processResponse()*.

Como se observa en este método (ver Ilustración 42) tenemos una gran variedad de caminos que puede seguir el flujo de ejecución. No obstante podemos agruparlo en tres partes:

- **Respuestas referentes al registro del usuario SIP**: como se puede observar en el diagrama de secuencia **sd UserRegister** se distinguen entre distintas respuestas del servidor. Cuando no se puede realizar el registro del usuario se le comunica al cliente, en caso contrario cuando todo ha funcionado correctamente el siguiente paso es el envío de una petición de sesión al Asistente. Además en este segundo caso se inicia un bucle que mantiene activo el registro del usuario en el servidor.
- **Respuesta a una petición *INVITE***: existen distintos tipos de respuesta para esta petición que deben tenerse en cuenta.
    - Respuestas de transición: son aquellas respuestas que indica se está pendiente de la aceptación de la sesión, no obstante confirman la conexión con la parte Asistente.
    - Respuesta negativa: es aquella respuesta generada cuando se declina la sesión (*DECLINE*).
    - Respuesta positiva: inicia la comunicación multimedia.
- **Respuestas "negativas"**: respuestas que implican por distintos motivos la imposibilidad de establecer la comunicación con el Asistente.

Igual que en su clase homónima *Assistance* de la parte Asistente, como bien se puede observar a lo largo de este punto, *Kiosc* funciona como el *SipListener* presentado en el apartado 3.3. No obstante, así como el contenido SDP para la parte Asistente se encuentra en la clase *Call*, para la del Kiosco esta información está presente en la clase

Kiosc. Esto se debe a la singularidad de la llamada en el Kiosco y a la propia simplicidad de la solución elaborada.

6.5 Pruebas

Para verificar y validar el correcto funcionamiento de nuestro AsistenteVoIP se realizaron distintos tipos de pruebas software. Las prueba software puede definirse como una actividad en la cual un sistema se ejecuta dentro de unas circunstancias preestablecidas, los resultados son observados y registrados para su posterior evaluación. Dichas pruebas software deben ir siempre orientadas a ofrecer un resultado para un objetivo específico (D Everett & McLeod, 2007).

Se llevaron a cabo pruebas de unidad que testeaban cada clase de una forma aislada del resto de componentes en un contexto dado. Con las funcionalidades y capacidades básicas ya comprobadas se pasó a la realización de las pruebas de integración, para comprobar si se cumplían los requisitos y no existían errores/fallos/defectos. Durante estas pruebas, cambiantes a la par que el sistema, se encontraron distintos puntos a mejorar y problemas que solucionar. Seguidamente presentaremos las pruebas más importantes realizadas.

6.5.1 Protocolo SIP y NAT

El protocolo SIP dentro de las redes locales (LAN) funciona correctamente sin ningún tipo de problema, por su contra cuando la comunicación viaja a través de Internet existe un problema; el *Network Address Translation* (NAT). NAT es una tecnología comúnmente utilizada por *firewalls* y *routers* para permitir a múltiples dispositivos de una LAN compartir una IP pública, el tipo de NAT más utilizado es la sobrecarga: se realiza un mapeo de la dirección a partir con un puerto aleatorio. Con el fin de ofrecer a un dispositivo de la red establecer contacto con otro dispositivo fuera de esta LAN la tecnología NAT establece un cambio entre la IP privada y al pública. De este modo una IP privada y un puerto dado pasarían a ser una IP pública y un puerto distinto:

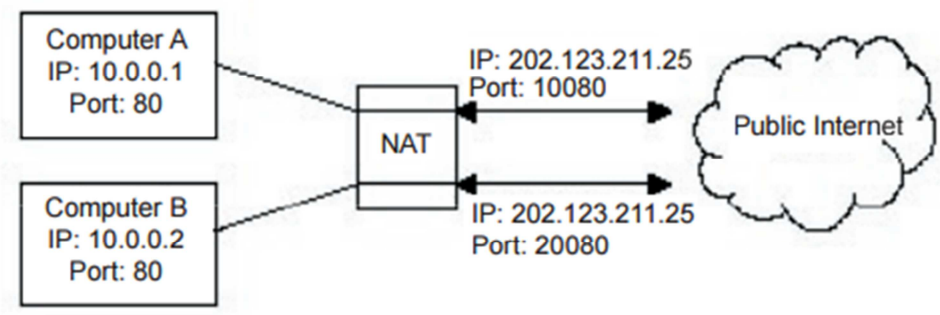


Ilustración 43: Funcionamiento NAT.

En el contenido SDP del protocolo SIP observamos que se indican los puertos seleccionados por cada parte para el envío y recepción de audio y video.

```
INVITE sip:10000@XX.XX.XX.XX:5060 SIP/2.0
Call-ID: 470a1874254eda10f97cf99c1b0b5fbc@192.168.80.83
CSeq: 1 INVITE
From: "1:2" <sip:90003@192.168.80.83>;tag=12345
To: "10000" <sip:10000@ XX.XX.XX.XX:5060>
Via: SIP/2.0/UDP XX.XX.XX.XX:5060;branch=z9hG4bK-313237-9afbf94b5bb1587e631a4ac487a0716d
Max-Forwards: 70
Contact: "1:2" <sip:90003@192.168.80.83:5060>
Supported: timer
Session-Expires: 4000
Min-SE: 4000
Content-Type: application/sdp
Content-Length: 293

v=0
o=servidor_video_1 2808844564 2808844564 IN IP4 192.168.80.83
s=
c=IN IP4 192.168.80.83
t=0 0
m=video 5100 RTP/AVP 99
a=rtpmap:99 H264/90000
m=audio 5000 RTP/AVP 3
a=rtpmap:3 GSM/8000
a=sendrecv
```

Ilustración 44: Respuesta a petición *INVITE* de la parte Asistente.

En la Ilustración 45 se puede observar el contenido SDP generado en la respuesta a la petición *INVITE* por la parte Asistente. Marcado en color rojo aparecen los puertos seleccionados para el envío de video y de audio respectivamente. En color azul se observa la dirección IP privada desde la que se está realizando la comunicación. Como hemos hablado previamente, un dirección privada y un puerto pasan a tener una IP pública (normalmente la misma para todos los dispositivos de la LAN) y un puerto aleatorio.

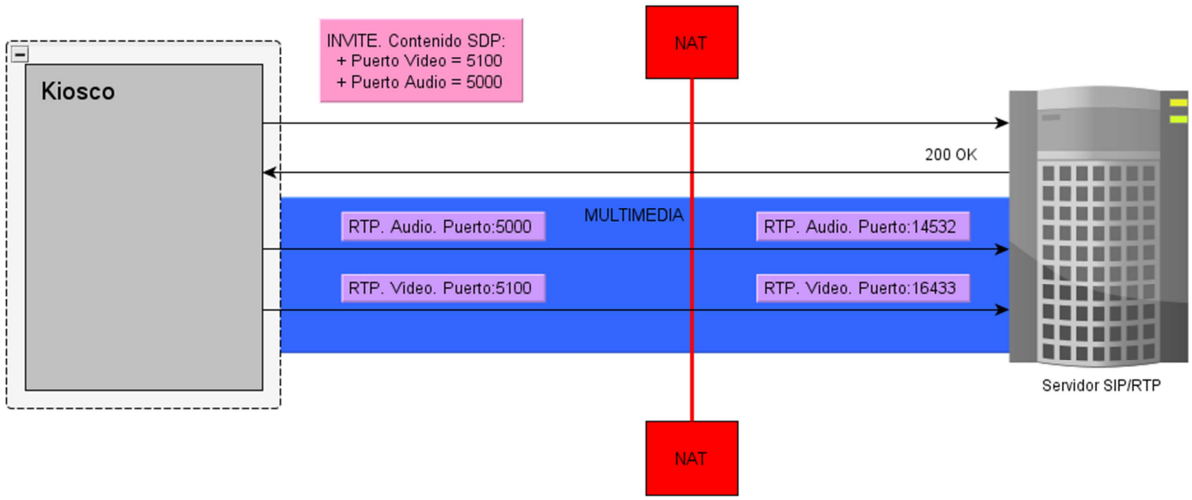


Ilustración 45: Problema NAT.

Por tanto el problema reside en la incongruencia entre puertos indicados que se van a utilizar para el envío y recepción de contenido multimedia y los puertos que se utilizan realmente debido al cambio realizado por la NAT. Existen diversas soluciones como son el protocolo STUN (implementado y testeado pero con resultados no deseados) y el ICE. No obstante se encontró la solución en el servidor Asterisk. Gracias a la

modificación de los parámetros de configuración, que podemos observar en el anexo A, se solucionaron los problemas relacionados con la NAT.

6.5.2 Registro SIP

Otra de las pruebas a destacar son las realizadas para asegurar el correcto **registro de un usuario SIP** en Asterisk, no tanto por su dificultad de testeo sino por su importancia en la aplicación. Como podemos observar en la Ilustración 25, este proceso requiere de dos peticiones y un cifrado de la contraseña del usuario, ambas funcionalidades testeadas en las pruebas de unidad con un resultado positivo, así tan solo era necesario realizar la prueba del conjunto. Por tanto, a pesar de tratarse de una prueba sencilla (la respuesta del servidor indica el propio resultado de esta) presentamos esta prueba debido a que se trata de un aspecto importantísimo para conseguir el funcionamiento completo de nuestro AsistenteVoIP.

6.5.3 Finalización sesiones SIP

También comentar aquellas pruebas relacionadas con el establecimiento y terminación de sesiones multimedia mediante el protocolo SIP y el SDP. Para realizar las pruebas dividimos estas en dos sub-conjuntos: pruebas **referentes al canal de comunicación** (pruebas SIP) y pruebas **referentes al contenido multimedia** (SDP). Por tanto primero se realizaron pruebas asegurando el correcto establecimiento y finalización de las sesiones (también el rechazo de estas) analizando las respuestas recibidas por el servidor y observando en este los registro de eventos. Aparecieron problemas durante la finalización de sesiones SIP mediante método *BYE* debido a la caducidad de la sesión, no obstante el problema se solucionó mediante el envío de peticiones *INVITE* en intervalos de tiempo.

Posteriormente se comprobó que el canal de comunicación se estableció correctamente a través del servidor Asterisk y que ambas partes pudiesen enviar y recibir audio y video.

6.5.4 Funcionalidades

Se realizaron extensas pruebas para asegurar el correcto funcionamiento de las funcionalidades implementadas de forma exhaustiva en cada iteración del proyecto, tanto de funcionalidades ya existentes en iteraciones anteriores como de nuevas.

6.5.5 Pruebas del sistema

Para concluir este apartado debemos hablar de las **pruebas del sistema en su totalidad**. Es decir, una vez testeadas todas las funcionalidades de nuestro sistema se realizaron diversas pruebas con el entorno de ejecución final. En un Kiosco se instaló su ejecutable correspondiente y en el PC del asistente se realizó lo propio. De este modo se mejoraron los aspectos conflictivos y posteriormente se comprobó el correcto funcionamiento del sistema en su entorno final de ejecución.

7. Conclusiones y trabajos futuros

En este capítulo presentaremos las conclusiones obtenidas, tanto a nivel personal como académico. También hablaremos sobre los posibles futuros trabajos que se pueden llevar a cabo para mejorar o reutilizar nuestra aplicación.

7.1 Conclusiones

En líneas generales podemos definir nuestro proyecto AsistenteVoIP como un proyecto práctico y funcional en su contexto, un módulo de una aplicación ya existente (TeKio) para la empresa Tecatel S.L. Un **proyecto** que nace de la necesidad de ofrecer asistencia en tiempo real para usuarios inexpertos y que gracias a un análisis del contexto tecnológico ofrece una solución actual a un problema moderno. Gracias también a los **requisitos** establecidos pudimos crear las bases de aquello que realmente nuestra aplicación debía ofrecer y los aspectos no funcionales que a cumplir. El **modelado** proporcionó una imagen de la estructura y la comunicación interna que debía seguir cada parte de nuestro sistema y a la vez proporcionó un medio de comunicación con los directivos de la empresa para mostrarles cómo se estructuraría el sistema.

Como hemos observado a lo largo del proyecto podemos concluir que se han cumplido los objetivos y se ha desarrollado un complemento a TeKio y que:

- **Ampliación de la implementación de los protocolos SIP y SDP:** aunque se han implementado las funcionalidades más básicas de ambos protocolos, la ampliación de más capacidades sería un punto a favor de nuestro asistente. Mejorando el soporte de estos protocolos se favorecería la capacidad de interacción con otros sistemas que los implementen.
- **Refactorización del código y guías de estilo:** el *refactoring* realizado de forma continua sobre el código de nuestro AsistenteVoIP ha mejorado de forma significativa el mantenimiento y entendimiento del sistema. Por otra parte, las guías de estilo han sido de gran ayuda para la estructura empleada en la programación.
- **Importancia y mejora del diseño:** el diseño ha supuesto un eje en el desarrollo de nuestra aplicación. Se empezó con un diseño muy básico, donde las clases eran pocas y de funcionalidades reducidas, no obstante, a medida que empezamos a entender el funcionamiento de los protocolos y las tecnologías utilizadas el diseño de nuestro sistema mejoró notablemente. Evolucionamos los diagramas de clases, las interacciones entre estas y los estados de las llamadas recibidas hasta el estado actual.

  Aun así, hemos de decir que en nuestro modelado existen puntos que sin duda serían mejorados en una próxima versión. Por tanto, con toda la experiencia adquirida, podemos comentar que el modelado supone un punto base a evolucionar, a la par de las necesidades y los objetivos.

- **Importancia de las pruebas:** aun pareciendo un aspecto trivial, la realización de pruebas sobre el sistema ha resultado un aspecto fundamental. Gracias a los test realizados se encontraron fallos/errores/problemas tanto a niveles de comunicación entre partes como para cada ejecutable. Además también se descubrieron problemas "externos" a nuestro sistema, como observamos en el apartado 6.3.

Personalmente creo que se trata de un proyecto que aporta una funcionalidad importante a un sistema y al que se le pueden añadir otras características para enriquecerlo aún más. No obstante, hay puntos que no han terminado de convencerme. A pesar que el protocolo SIP ofrece una gran variedad de capacidades, existen aspectos que le afectan negativamente como es el caso de la NAT. Por otro lado el protocolo SDP aporta, sin ningún tipo de problema, el manejo de sesiones multimedia. Las funcionalidades ofrecidas por subversión sumadas a su integración en Eclipse y su facilidad de uso han sido de gran ayuda para controlar las versiones. La librería LibJitsi nos ha servido para la comunicación multimedia, no obstante he podido apreciar que ofrece una gran variedad de opciones que actualmente no se han utilizado. LibJitsi es una librería realmente completa.

En definitiva, hemos realizado un proyecto donde se han cumplido los objetivos marcados respetando los requisitos establecidos. Hemos aprendido a valorar la importancia del diseño correcto de una aplicación para evitar futuros problemas, mantener la independencia entre módulos, mejorar la entendibilidad, mejorar la calidad... Hemos estudiado y analizado dos protocolos muy interesantes. También hemos aprendido a valorar la importancia de las pruebas para encontrar aspectos a corregir. Es decir, hemos realizado una aplicación funcional adecuada a su contexto y hemos aprendido durante su proceso de desarrollo.

7.2 Trabajos futuros

Partiendo de una primera versión operativa en la que se cumplen los objetivos establecidos, procedemos a ofrecer distintos caminos en los que puede evolucionar ésta. Todos ellos teniendo siempre presente que el objetivo es mejorar la interacción con el cliente pero sin renunciar a un correcto desarrollo:

- Como hemos comentado, Tecatel dispone de terminales de reproducción de contenido bajo demanda en distintos centros hospitalarios. En cada uno de estos recintos se dispone de un empleado encargado de resolver las dudas y los posibles problemas de los usuarios, no obstante puede ser que este no esté disponible en cierto momento. Sabiendo que los terminales tienen una implementación software que les permite establecer comunicación de audio mediante el protocolo SIP, se podría crear una nueva versión del **software del asistente para que tuviera la capacidad de comunicarse con dichos terminales**. De este modo en aquellos intervalos de tiempo en los que el empleado de Tecatel no esté disponible el usuario tendría la posibilidad de comunicarse con un asistente remoto vía SIP.
- Incorporar un **contestador automático** donde guardar los mensajes de los usuarios cuando sus llamadas no han podido ser atendidas y posteriormente la

recuperación y reproducción de éstos. Gracias al equipo de desarrollo de Tecatel S.L sabemos que el servidor Asterisk ofrece la posibilidad de disponer de un contestador para cada usuario SIP definido, por tanto verdaderamente tan solo deberíamos desarrollar en nuestra aplicación la recuperación y reproducción de los mensajes.

- Desarrollar **mejoras en el almacenamiento de datos destacables**. Dicho de otro modo, mejorando la cantidad y la calidad de los datos almacenados se podrían realizar estudios para mejorar el nivel de atención al usuario. Guardando datos como podrían ser los estados por los que ha pasado una llamada, el trabajador que le atendió, tiempo hasta atender una llamada... También se podrían desarrollar diferentes tipos de encuestas de satisfacción para ser respondidas por parte del usuario al finalizar la llamada. En definitiva, realizar una amplia mejora de la calidad y cantidad de los datos guardados para poder mejorar la **ingeniería social** a realizar.
- Integrar un **chat** tanto en la parte kiosco como en la asistente, de este modo en aquellos escenarios donde se disponga una conexión a internet que no permita el envío ni de audio ni de video, se dispondría de un tercer módulo capaz de comunicar el asistente y el usuario.
- Como cuarta vía de posible desarrollo aparece el acoplamiento de nuestra parte asistente a un **Applet** de JAVA para poder ejecutar nuestro asistente en entorno web y así evitar la dependencia del archivo .jar en el sistema. No obstante esta vía aún está pendiente de ser confirmada su viabilidad debido a las restricciones que aparecen en el entorno de ejecución de una Applet mediante el navegador. Además también se debería de estudiar el soporte futuro de los navegadores web a esta tecnología.
- Por último, y siendo conscientes de las dificultades, tenemos el desarrollo de un módulo que permita al **asistente realizar acciones sobre el Kiosco** mediante la interfaz gráfica de este.

Bibliografía

Bryant, R., Madsen, L., & Van Meggelen, J. (2013). *Asterisk: The Definitive Guide* (Cuarta ed.). O'Reilly Media.

Bass, L., Clements, P., & Kazman, R. (1998). *Software Architecture In Practice*. Addison-Wesley.

Booch, G., Jacobson, I., & Rumbaugh, J. (2006). *El lenguaje unificado de modelado. UML 2.0 2ªEdición*. Addison-Wesley.

Bruegge, B., & H. Dutoit, A. (2009). *Object-Oriented Software Engineering Using UML, Patterns, and Java*. Prentice Hall.

Burnstein, I. (2003). *Practical Software Testing: A Process-Oriented Approach*. Springer.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-Oriented Software Architecture*. John Willey & Sons.

Collins-Sussman, B., W. Fitzpatrick, B., & Pilato, M. (2008). *Version Control with Subversion: For Subversion 1.7*.

D Everett, G., & McLeod, R. (2007). *Software Testing: Testing Across the Entire Software Development Life Cycle*. IEEE PRESS.

Dennis Baron. (2015). *http://web.mit.edu/*. Obtenido de http://web.mit.edu/sip/presentations/np119.pdf

DuBois, P. (2013). *MySQL (Developer's Library)* (Quinta ed.). Addison-Wesley Professional.

Eclipse. (2013). *Eclipse Modeling Framework Project (EMF)*.

Google. (s.f.). *google-styleguide.googlecode.com*. Obtenido de https://google-styleguide.googlecode.com/svn/trunk/javaguide.html

H. Kan, S. (2002). *Metrics and Models in Software Quality Engineering* (Segunda ed.). Addison Wesley.

Handley, M., & Jacobson, V. (July 2006). *SDP: Session Description Protocol (RFC4566)*. Universidad de Glasgow.

IBM. (2015). *ibm.com*. Obtenido de http://www.ibm.com/developerworks/rational/library/3101.html

in2eps. (2014). *into the 3GPP Evolved Packet System*. Obtenido de http://www.in2eps.com/fo-sip/tk-fo-sip-ex3261.html

Ingate® Systems. (s.f.). *Solving the Firewall/NAT Traversal Issue of SIP*.

International Organization for Standardization. (2008). *ISO 9001*.

Jacobson, I., Christerson, M., Jonsson, P., & Gunnar, O. (1992). *Object-Oriented Software Engineering, A Use Case Driven Approach*. Addison-Wesley.

Johnston, A. B. (2003). *Understandig the Session Initiation Protocol (2 Edition)*. Artech House Inc.

Mishra, S. (2008). *Visual Modeling & Unified Modeling Language (UML) : Introduction to UML*. Rational Software Corporation.

Niemeyer, P., & Knudsen, J. (2013). *Learning Java* (Cuarta ed.). O'Reilly Media.

OMG. (09 de 10 de 2014). Obtenido de http://www.omg.org/

Oracle. (s.f.). *jsip.java.net*. Obtenido de https://jsip.java.net/

Rational Software. (2011). *Rational Unified Proces: Best Practices for Software Development Teams*.

S. Pressman, R. (2010). *Ingeniería del software. Un enfoque práctico*. (Tercera ed.). McGRAW-HILL.

Sierra, K., & Bates, B. (2005). *Head First Java* (Segunda ed.). O'Reilly Media.

Sommerville, I. (2005). *Ingenierá del software* (Séptima ed.). Addison-Wesley.

Sterman, B., & Schwartz, D. (s.f.). *NAT Traversal in SIP*. deltathreee: The IP Communications Network.

Stevens, P., & Pooley, R. (2008). *Utilización de UML en Ingeniería del Software con Objetos y Componentes* (Segunda ed.). Addison-Wesley.

Sznajdleder, P. A. (2013). *Java a fondo. Estudio del lenguaje y desarrollo de aplicaciones*. (Segunda ed.). Alfaomega.

UPV. (2014). Apuntes Ingenieria del Software.

voip-info.org, & E. Johansson, O. (2012). *voip-info.org*. Obtenido de http://www.voip-info.org/wiki/view/Asterisk+introduction

Weitzenfeld, A. (2005). *Ingeniería del Software OO con UML. Java e Internet*. Thomson.

Anexos

A. Configuración del entorno de desarrollo.
B. Casos de uso.
C. Diagramas secuencia.

Anexo A

Configuración de la aplicación

1. Ejecutables.

La aplicación desarrollada está separada en dos archivos ejecutables de JAVA (.jar). Cada archivo corresponde a cada una de las partes que intervienen en la comunicación. Además dispondremos de distintas versiones de los ejecutables (32 i 64 bits). Para identificar cada ejecutable debemos acceder a la carpeta "AsistenteVoIP", allí tendremos la siguiente estructura:

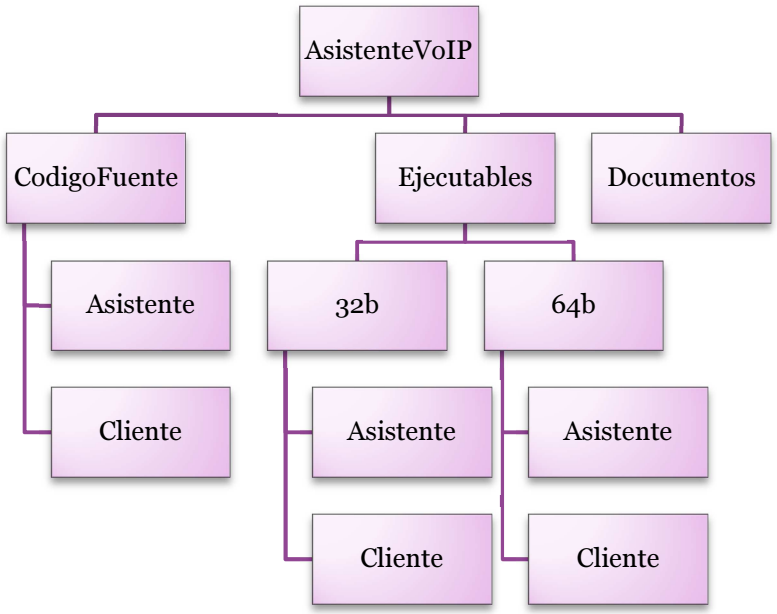


Ilustración 46: Esquema de la carpeta raíz.

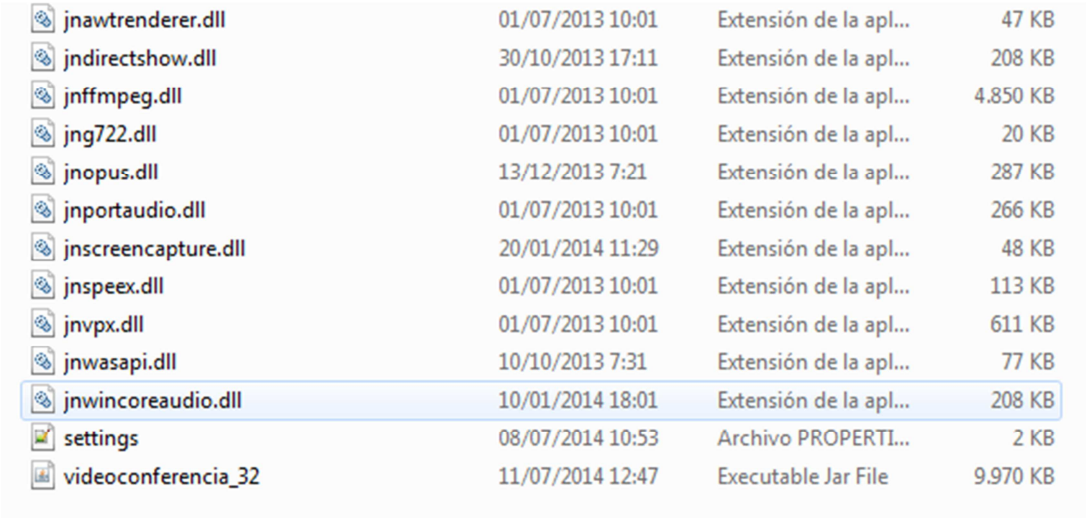
Como se observa en la figura disponemos de distintas versiones de asistente y cliente en las carpetas que identifican dichas versiones.

2. Instalación.

Para tener el software instalado tan solo es necesario copiar los archivos, del último directorio que identifica a la versión, en el directorio deseado. Así los archivos que nos encontraremos son comentados a continuación.

2.1 Asistente

Los archivos disponibles y que se deben copiar son los siguientes:



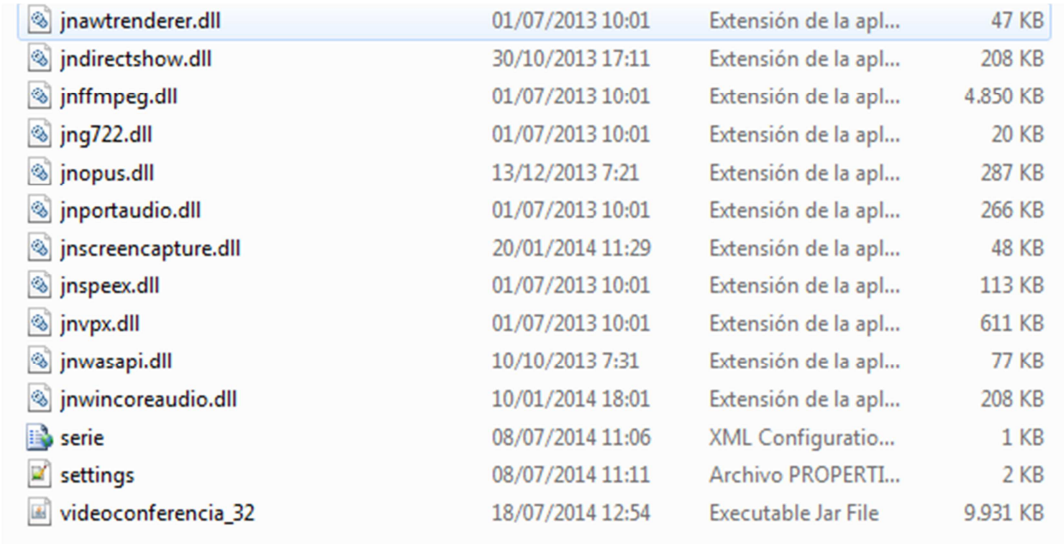
jnawtrenderer.dll	01/07/2013 10:01	Extensión de la apl...	47 KB
jndirectshow.dll	30/10/2013 17:11	Extensión de la apl...	208 KB
jnffmpeg.dll	01/07/2013 10:01	Extensión de la apl...	4.850 KB
jng722.dll	01/07/2013 10:01	Extensión de la apl...	20 KB
jnopus.dll	13/12/2013 7:21	Extensión de la apl...	287 KB
jnportaudio.dll	01/07/2013 10:01	Extensión de la apl...	266 KB
jnscreencapture.dll	20/01/2014 11:29	Extensión de la apl...	48 KB
jnspeex.dll	01/07/2013 10:01	Extensión de la apl...	113 KB
jnvpx.dll	01/07/2013 10:01	Extensión de la apl...	611 KB
jnwasapi.dll	10/10/2013 7:31	Extensión de la apl...	77 KB
jnwincoreaudio.dll	10/01/2014 18:01	Extensión de la apl...	208 KB
settings	08/07/2014 10:53	Archivo PROPERTI...	2 KB
videoconferencia_32	11/07/2014 12:47	Executable Jar File	9.970 KB

Ilustración 47: Archivos a copiar para la parte Asistente.

Como se observa en la imagen se dispone de varios archivos *.dll, estos archivos son necesarios para el envío y recepción de audio y **siempre** deben estar en el mismo directorio que el ejecutable. Dicho ejecutable posee el nombre de "videoconferencia_(nºbits)" y es la esencia de la aplicación. Por último se dispone de un fichero de configuración desde donde se pueden modificar distintos parámetros.

2.2 Cliente

Los archivos disponibles para la parte Kiosco y que se deben copiar son los siguientes:



jnawtrenderer.dll	01/07/2013 10:01	Extensión de la apl...	47 KB
jndirectshow.dll	30/10/2013 17:11	Extensión de la apl...	208 KB
jnffmpeg.dll	01/07/2013 10:01	Extensión de la apl...	4.850 KB
jng722.dll	01/07/2013 10:01	Extensión de la apl...	20 KB
jnopus.dll	13/12/2013 7:21	Extensión de la apl...	287 KB
jnportaudio.dll	01/07/2013 10:01	Extensión de la apl...	266 KB
jnscreencapture.dll	20/01/2014 11:29	Extensión de la apl...	48 KB
jnspeex.dll	01/07/2013 10:01	Extensión de la apl...	113 KB
jnvpx.dll	01/07/2013 10:01	Extensión de la apl...	611 KB
jnwasapi.dll	10/10/2013 7:31	Extensión de la apl...	77 KB
jnwincoreaudio.dll	10/01/2014 18:01	Extensión de la apl...	208 KB
serie	08/07/2014 11:06	XML Configuratio...	1 KB
settings	08/07/2014 11:11	Archivo PROPERTI...	2 KB
videoconferencia_32	18/07/2014 12:54	Executable Jar File	9.931 KB

Ilustración 48:Archivos a copiar para la parte Kiosco.

Igual que la versión asistente los archivos *.dll son librerías y, como hemos dicho antes¸ **siempre** deben estar en el mismo directorio que el ejecutable. El ejecutable para la parte cliente posee las mismas características que la parte asistente. Esta parte también posee un archivo de configuración no obstante es distinto al del asistente.

3. Ejecutar aplicación.

Para ejecutar la aplicación y evitarnos posibles problemas la mejor opción es realizarlo desde la terminal Windows con la siguiente instrucción:

java –jar videoconferencia_32b-.jar

Debemos tener en cuenta que el ejecutable de la parte cliente será ejecutado por el propio sistema padre (kiosco) reaccionado a la pulsación del botón de asistencia. No obstante la parte del asistente requiere que el usuario ejecute la orden anterior para el arranque.

3.1 Cliente

En las versiones de los kioscos que disponen de la cámara Hercules HD Twist se precisa instalar previamente el software de la cámara. Una vez instalado hemos de desactivar el arranque por defecto de la aplicación proporcionada por la propia cámara.

4. Archivos de configuración.

Una vez explicados todo lo anterior pasaremos a explicar los archivos de configuración de cada parte.

4.1 Asistente

El fichero de configuración del asistente posee la siguiente estructura:

```
#Fichero de configuración
#Fri Jul 11 09:27:28 CEST 2014
localportaudio=6000
password=XXXXXXXX
audiodevice=Microphone (Realtek High Definition Audio) \:
stunserver=XX.XX.XX.XX
mysqlport=3306
localsipport=5060
stunport=3478
videodevice=ASUS USB2.0 WebCam \: directshow\:ASUS USB2.0
server=XX.XX.XX.XX
stun=false
mysqlpassword=XXXXXX
localportvideo=7000
mysqldb=XXXXXX
mysqlserver=XX.XX.XX.XX
serversipport=5060
refreshregistrationtime=80000
mysqluser=videoconferencia
user=10000
savecalls=true
```

Ilustración 49: Fichero de configuración para la parte Asistente.

Vamos a explicar los distintos parámetros agrupándolos por familias.

4.1.1 SIP/SDP

Línea	Parámetro	Explicación
4	password	Contraseña del usuario SIP en el servidor Asterisk.
8	localsipport	Puerto local de interacción del protocolo SIP.
11	server	Dirección IP del servidor SIP (Asterisk).
17	serversipport	Puerto SIP del servidor.
18	refreshregistrationtime	Tiempo de refresco del registro para mantener al asistente alcanzable.
20	user	Usuario asistente a registrar.

4.1.2 STUN

Línea	Parámetro	Explicación
6	stunserver	Dirección IP del servidor STUN.
9	stunport	Puerto del servidor STUN.
12	stun	*Boolean* (*true* o *false*) que indica si se realiza el cálculo de la IP y los puertos públicos.

4.1.3 MySQL

Estos parámetros nos sirven para que la aplicación sea capaz de ser manejada desde una base de datos MySQL.

Línea	Parámetro	Explicación
7	mysqlport	Puerto del servidor MySQL.
13	mysqlpassword	Contraseña del usuario en el servidor MySQL.
15	mysqldb	Nombre de la base de datos del servidor MySQL.
16	mysqlserver	Dirección IP del servidor MySQL.
19	mysqluser	Usuario del servidor MySQL.
21	savecalls	Indica si se quiere guardar la información importante de las llamadas en la base de datos MySQL.

4.1.4 Configuraciones locales

Línea	Parámetro	Explicación
3	localportaudio	Puerto desde donde se enviará y recibirá el audio.
5	audiodevice	Cadena de caracteres identificativa del dispositivo de audio predeterminado. (Tiene importancia cuando no se utiliza IGU). **Solo modificable por la aplicación**
10	videodevice	Cadena de caracteres identificativa del dispositivo de video predeterminado. (Tiene importancia cuando no se utiliza IGU).

		Solo modificable por la aplicación
14	localportvideo	Puerto desde donde se enviará y recibirá el video.

4.2 Cliente.

El fichero de configuración del Kiosco posee la siguiente estructura:

```
1  #Fichero de configuración
2  #Thu Jul 10 13:06:08 CEST 2014
3  remoteDataFromDB=true
4  localportaudio=6098
5  password=XXXXXXX
6  audiodevice=Microphone (Realtek High Definition Audio) \: 
7  mysqlport=3306
8  localsipport=5060
9  kiosco=2
10 toserver=XX.XX.XX.XX
11 fromuser=90003
12 audiocode=3
13 hospital=1
14 videodevice=ASUS USB2.0 WebCam \: directshow\:ASUS USB2.0 
15 touser=10000
16 localportvideo=6100
17 mysqldb=XXXXXX
18 mysqlserver=XX.XX.XX.XX
19 serversipport=5060
20 refreshregistrationtime=80000
```

Ilustración 50: Fichero de configuración para la parte Kiosco.

Igual que en el apartado anterior procedemos a la descripción por familias:

4.2.1 SIP/SDP

Línea	Parámetro	Explicación
5	password	Contraseña del usuario SIP en el servidor Asterisk.
8	localsipport	Puerto local de interacción del protocolo SIP.
10	toserver	Dirección IP del servidor SIP (asterisk).
11	fromuser	Usuario SIP origen de la comunicación entre el asistente y el cliente.
15	touser	Usuario SIP destino de la llamada.
19	serversipport	Puerto SIP del servidor.
20	refreshregistrationtime	Tiempo de refresco para mantener al usuario registrado en el servidor, para así poder ser alcanzable.

4.2.2 MySQL

Línea	Parámetro	Explicación
3	remoteDataFromDB	*Boolean* (*true* o *false*) que indica si los datos SIP a utilizar por la aplicación se leen desde fichero o desde base de datos. En caso de que se lean de la base de datos (*true*) el fichero posteriormente será actualizado.
7	mysqlport	Puerto del servidor MySQL.
17	mysqldb	Nombre de la base de datos del servidor MySQL desde donde se leerán los datos nuevos.
18	mysqlserver	Dirección IP del servidor MySQL.

4.2.3 Configuraciones locales

Línea	Parámetro	Explicación
4	localportaudio	Puerto desde donde se enviará y recibirá el audio.
6	audiodevice	Cadena de caracteres identificativa del dispositivo de audio predeterminado. En la parte cliente el dispositivo no puede ser seleccionado por el usuario sino por un técnico. **Solo modificable por la aplicación.**
9	kiosco	Número identificador del kiosco.
12	audiocode	Numero identificador del code de audio a utilizar por la aplicación. Tan solo se permiten tres: 0 → PCMU 3 → GSM 8 → PCMA
13	hospital	Número identificador del hospital.
14	videodevice	Cadena de caracteres identificativa del dispositivo de video predeterminado. En la parte cliente el dispositivo no puede ser seleccionado por el usuario sino por un técnico. **Solo modificable por la aplicación**
16	localportvideo	Puerto desde donde se enviará y recibirá el video.

Como se observa esta parte no dispone de la unidad familiar STUN debido a que estudios en la parte asistente demostraron que el uso del STUN no solucionaba ningún problema en el entorno final de la aplicación.

5. Primera ejecución.

Cuando se ejecute por primera vez la aplicación cliente se debe asegurar que los parámetros de configuración sean los correctos, a pesar de que la aplicación tiene la capacidad de comprobar la validez estructural de los datos hay campos que no se pueden comprobar (usuario, contraseña, correcta dirección IP…). Con la primera ejecución de la aplicación esta busca en el sistema todos los aparatos multimedia compatibles y que sean capaces de reproducir y/o capturar audio y/o video. Por tanto el técnico debe asegurarse de realizar la selección de los dispositivos en la ventana que saldrá en la primera ejecución:

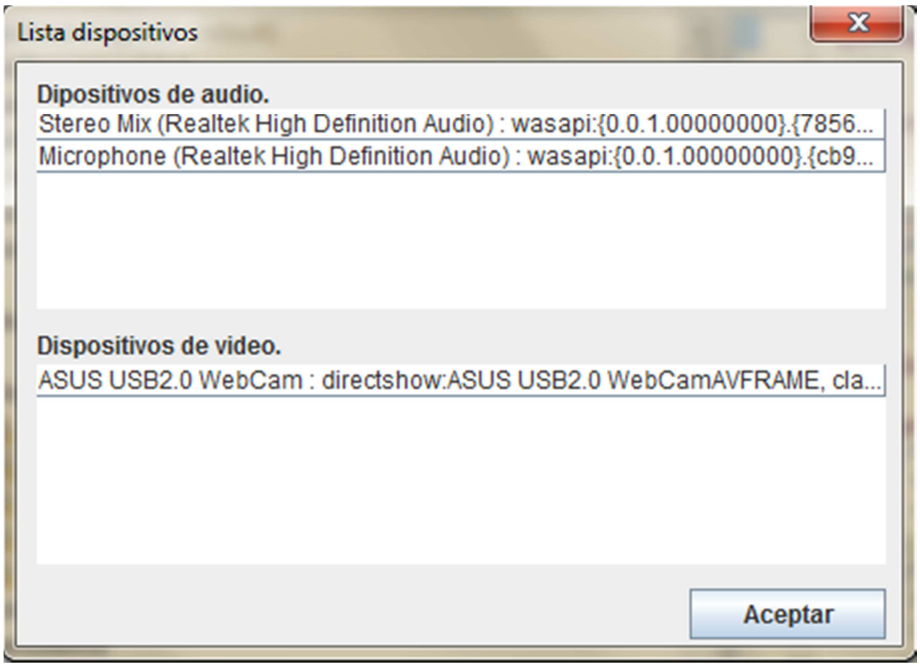


Ilustración 51: Primer inicio de la parte Kiosco.

En esta ventana se seleccionan los dispositivos predeterminados a utilizar. En el caso que se quisiera cambiar los dispositivos es suficiente con eliminar el contenido a partir del "=" en el archivo de *settings.properties* para los parámetros *audiodevice* y/o *videodevice* y reiniciar la aplicación.

6. Proyecto eclipse.

Con el objetivo de mejorar y/o mantener el código fuente de la aplicación, en la carpeta "CodigoFuente" encontramos los proyectos eclipse tanto para la parte cliente como la asistente. Una vez importados los proyectos tendremos las siguientes estructuras.

6.1 Librería LibJitsi.

Para la parte cliente:

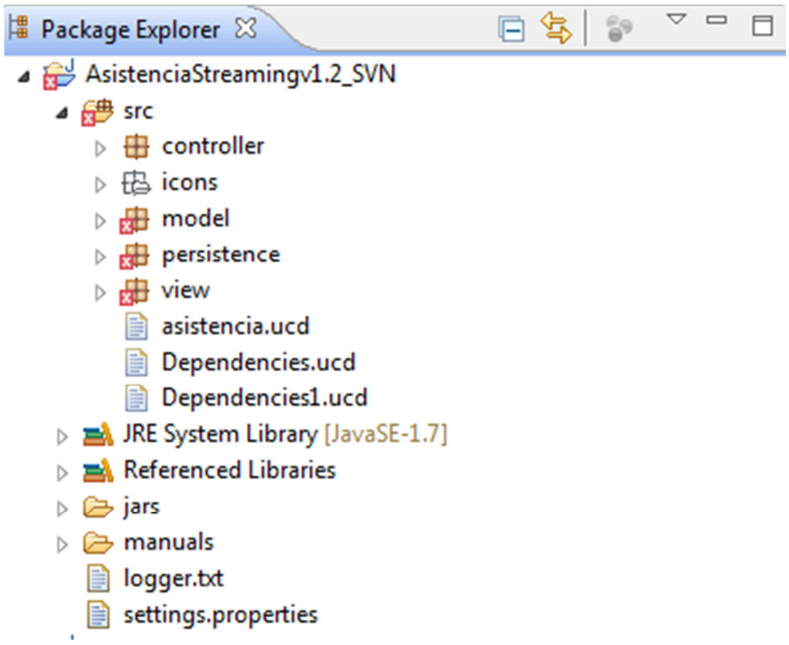


Ilustración 52: Estructura del proyecto eclipse.

Nos aparecerán una serie de errores, para solucionar dichos errores debemos crear la librería de usuario libjitsi en eclipse y agregarla al proyecto. Para ello seguimos estos pasos:

1. Accedemos a *Window -> Preferences -> Java -> Buid Path -> User Libraries*. Se nos mostrará una ventana similar a esta.

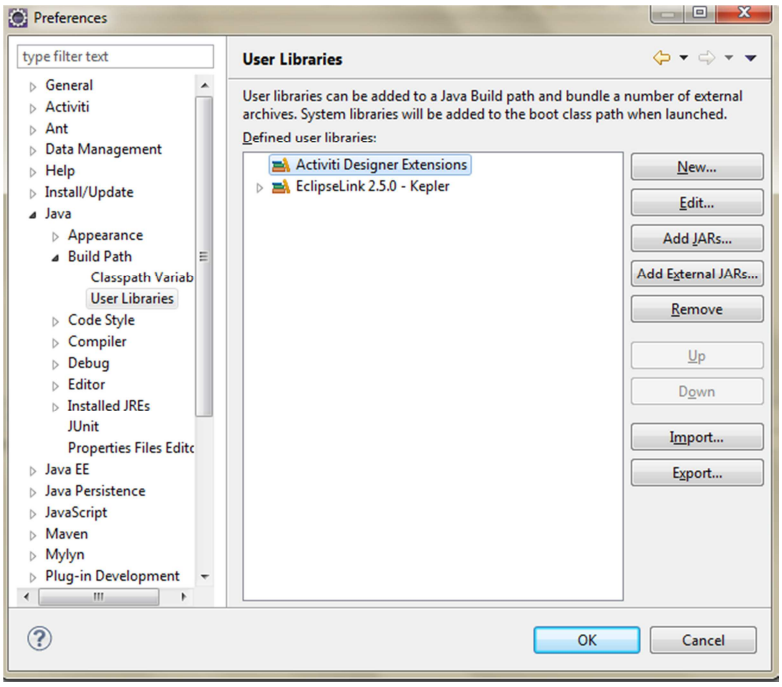


Ilustración 53: Ventana para añadir librerias al proyecto eclipse.

2. A continuación creamos la nueva librearía dándole a *New*. Crearemos dos librearías una de 64 y otra de 32b.
3. Una vez creadas las dos librerías tendremos lo siguiente:

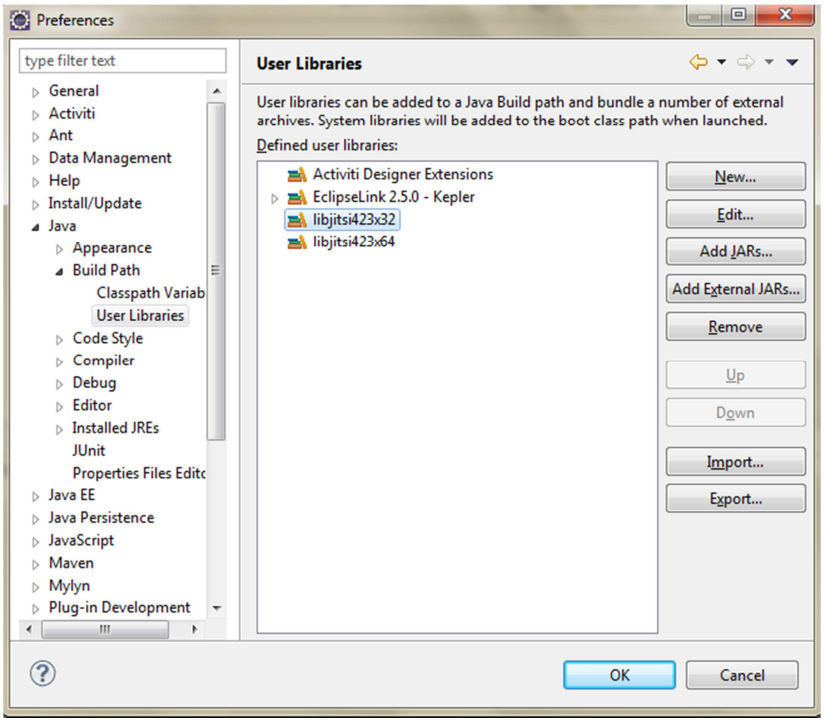


Ilustración 54: Ventana con las librerías ya añadidas.

Seguidamente debemos añadir los archivos fuente de la librería (debemos utilizar versiones iguales o superiores a la 423), dichos archivos los podemos descargar de: https://download.jitsi.org/libjitsi/ . Con los archivos descargados y descomprimidos donde queramos ya podemos agregarlos. Seleccionamos una librería en eclipse y pulsamos en *Add Externarls JARs*. Ahí debemos seleccionar el JAR que este en (DIRECTORIO LIBJITSI)/libjitsi. Nótese que para cada versión de la aplicación (32 o 64 bits) debemos seleccionar la librería correspondiente.

4. En este punto tendremos lo siguiente:

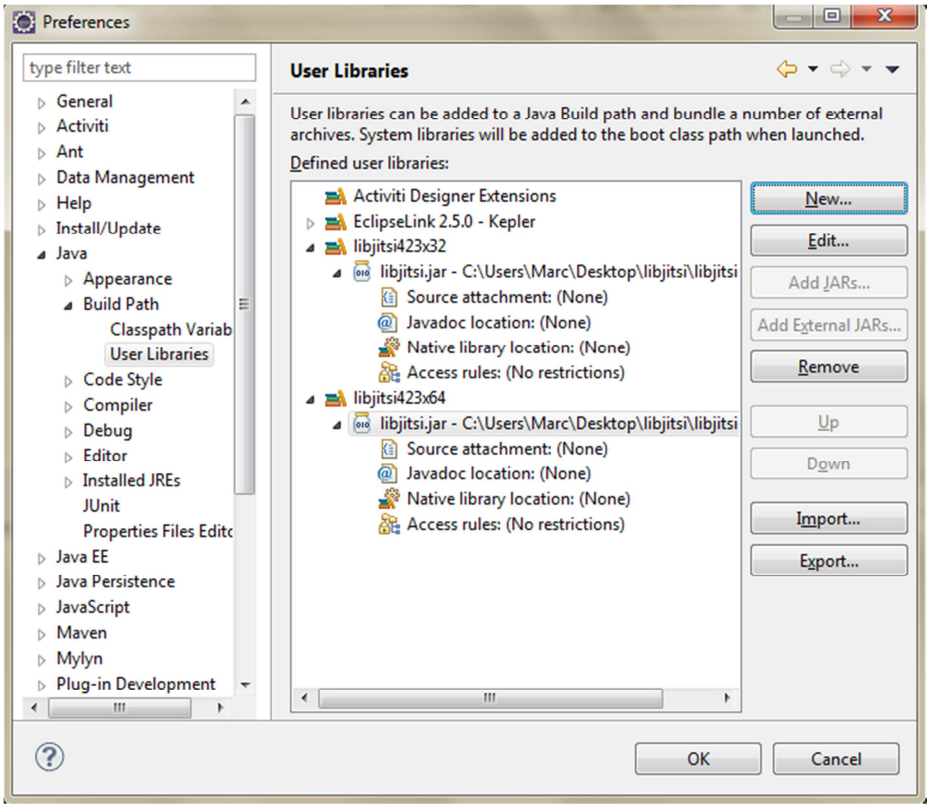


Ilustración 55: Ventana configuración de los archivos nativos para las librerias.

Ahora debemos añadir las librerías nativas *dll. Pulsando sobre *Native library location* con doble clic nos aparecerá una ventana desde donde seleccionar el directorio donde están los *dll (disponibles en DIRECTORIO_LIBJITSI/ lib\native\windows-(32 o 64)).

5. Pulsamos sobre OK y ya tenemos las librerías de 32 y 64 bits creadas. Dependiendo de la versión que queramos desarrollar seleccionaremos una librería u otra.
6. Por último pulsamos con el botón derecho sobre el proyecto -> *Build Path* -> *Configure Build Path*. Ahí podremos observar todas las librerías que ya dispone el proyecto, por defecto tendrá definidas algunas librerías para proyectos de 64bits*. A continuación pulsamos sobre *Add library* -> *User Library*, seleccionamos la librería a añadir y pulsamos sobre *Finish*.

*Nota: si deseamos cambiar las librerías debemos eliminarlas y agregar las nuevas desde PROYECTO/jars/. Seleccionamos los JARs que están en el mismo directorio y los de la librería libjitsi que queramos.

Agregada la librería ya nos aparecerá el proyecto sin errores:

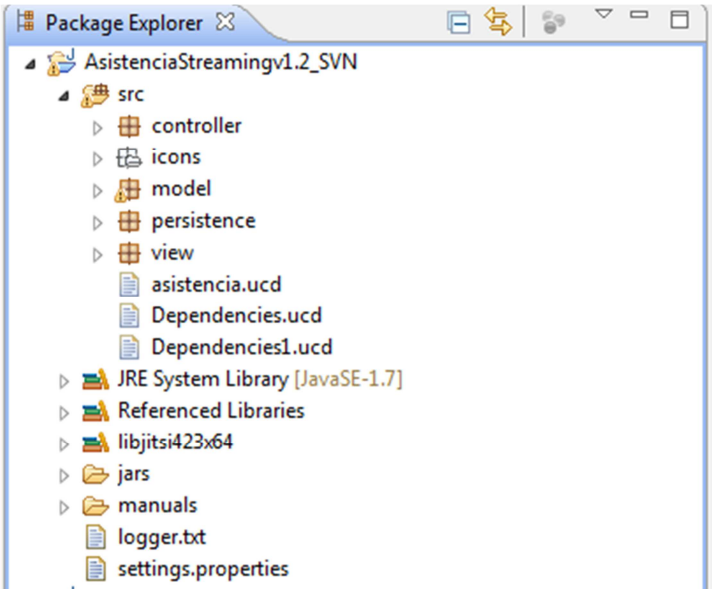


Ilustración 56: Esquema con la librería del proyecto eclipse.

6.2 Vista global.

A pesar que en la imagen 11 se observa el proyecto de la parte de la aplicación asistente (solo nos interesa la estructura) veremos los aspectos que son comunes en las dos partes.

En la carpeta *src* encontramos el código fuente de la aplicación dividida en cinco paquetes:

- **Controller**: contiene el controlador y se encarga de comunicar la interfaz con el modelo.
- **Icons:** contiene los iconos del programa.
- **Model:** lógica de la aplicación, interactua y modifica la interfaz.
- **View:** contiene las vistas de la aplicación.

Se ha utilizado la estructura *Model-View-Controller* debido a la propia naturaleza del problema. El siguiente campo que nos afecta es la carpeta *jars* que contiene todos los JARs (librerías) necesarias para ejecutar la aplicación tanto de 32 como de 64 bits.

Anexo B

Casos de uso

Parte Asistente

Caso de uso	06- Rechazar llamada.
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	La aplicación deberá comportarse tal como se describe en el siguiente caso de uso cuando el asistente rechace una llamada.
Precondiciones	Se ha realizado el caso de uso 01 y 16.
Postcondiciones	-
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El asistente decide rechazar una llamada entrante.	2. Envía una respuesta de rechazo al servidor SIP.
	3. Oculta la ventana de notificación de la llamada.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. Si en 2 no se puede contactar con el servidor después de varios intentos, la parte kiosco quedará sin notificar el rechazo de la petición. Se sigue el flujo normal para el rechazo de una llamada.
Comentarios	-

Caso de uso	07- Aceptar llamada.
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	La aplicación deberá comportarse tal como se describe en el siguiente caso de uso cuando el asistente acepte una llamada.
Precondiciones	Se ha realizado el caso de uso 01 y 16.
Postcondiciones	Se abre una ventana donde se observan las imágenes recibidas y enviadas. También se empieza a reproducir el audio recibido y a

	enviar el capturado.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El asistente acepta una llamada entrante.	2. Se leen los datos de comunicación para el establecimiento de una sesión SIP y los datos necesarios para establecer una sesión multimedia (SDP).
	3. Se envía una respuesta de aceptación al servidor SIP con los datos necesarios para la comunicación.
	4. Se inicia el envío y recepción de audio y video.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. En caso que en 3 no se pueda contactar con el servidor SIP no se inicia el envío y recepción de audio y se le notifica al asistente.
Comentarios	-

Caso de uso	**08- Registro asistente**
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	El asistente decide estar disponible para las llamadas provenientes de los Kioscos.
Precondiciones	-
Postcondiciones	-
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
	1. El sistema carga los datos necesarios para comunicarse con el servidor SIP.
	2. Se envían los mensajes necesarios al servidor para realizar el registro del usuario correspondiente al asistente.
	3. Se notifica al usuario el éxito del registro.
Extensiones síncronas	
Extensiones asíncronas	
	1. En 2 no se consigue contactar con el servidor debido a problemas de conexión, la aplicación notifica al asistente que el registro

	no ha sido posible y termina el caso de uso.
	2. En 2 Los datos correspondientes al asistente presente en los datos del servidor SIP no se corresponden con los datos que dispone la aplicación o por cualquier otro motivo no se puede realizar el registro, se le notifica al asistente la incapacidad de registro y termina el caso de uso.
Comentarios	-

Caso de uso	**09- Poner a la espera.**
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	En el momento en que se recibe una llamada el asistente decide pausar la llamada directamente, sin necesidad de ser atendida previamente.
Precondiciones	Se ha realizado los casos de uso 01 y 07.
Postcondiciones	La persona que ha realizado la llamada desde un kiosco deja de ser capaz de recibir audio y video.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El asistente decide poner una llamada a la espera si haber activado esta previamente.	2. Se envía un mensaje al servidor con destino el usuario que he empezado la llamada informando de la selección del asistente.
	3. Se inicia la recepción de audio solamente.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. En 2 no se consigue contactar con el servidor para el envío del mensaje, se inicia la recepción de video sin que la parte del kiosco sepa que está en espera y el caso de uso termina.
Comentarios	-

Caso de uso	**10- Finalizar llamada asistente.**
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	El asistente decide finalizar la comunicación con un Kiosco determinado.
Precondiciones	Se ha realizado el caso de uso 07.

Postcondiciones	Ya no existe ningún tipo de comunicación con el Kiosco llamante.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El decide finalizar una llamada determinada.	2. Se le comunica al servidor la decisión de finalizar la llamada, mediante los mensajes pertinentes.
	3. El sistema finaliza el envío y recepción de audio y video.
Extensiones síncronas	
	1. En 3 para el caso en que la llamada este pausada tan solo se finaliza la recepción de audio.
Extensiones asíncronas	
	1. En el paso 2 la aplicación no consigue contactar con el servidor, se finaliza la llamada en parte asistente y termina el caso de uso.
Comentarios	-

Caso de uso	**11-Pausar llamada asistente.**
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	Durante una llamada activa (no pausada) el asistente puede poner esta en pausa.
Precondiciones	Se ha realizado el caso de uso 07.
Postcondiciones	Se finaliza el envío de audio y video, y pasa a recibirse tan solo video.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El asistente pulsa sobre la opción de pausar una llamada.	2. Envía un mensaje indicando a la parte kiosco indicando que la llamada ha sido pausada.
	3. Se para el envío de audio y video.
	4. La parte asistente inicia la recepción de audio solamente.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. En 2 no se consigue contactar con el servidor para el envío del mensaje, se inicia la recepción de video sin que la parte del kiosco

	sepa que está pausada y el caso de uso termina.
Comentarios	La decisión de pausa de la llamada en la parte asistente implica un cambio en el estado de la llamada en la parte Kiosco. Así pues si el mensaje se perdiera o no fuera posible transferirlo a su destino, el asistente podría mandar más mensajes para modificar el estado.

Caso de uso	**12-Reiniciar llamada.**
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	Después de pausar una llamada el asistente de decide reiniciarla.
Precondiciones	Se ha realizado el caso de uso 07 o 09.
Postcondiciones	Se vuelve a iniciar la comunicación (visual y auditiva) entre usuario y asistente, del mismo modo que en caso de uso 07.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
El asistente pulsa sobre el botón para reiniciar la llamada pausada.	2. Se envía un mensaje indicando a la parte Kiosco que la llamada ha sido reiniciada.
	3. Se inicia la recepción y envío de audio y video.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. En 2 no se puede contactar con el servidor, no obstante se realizan toda la resta de acciones para la recepción y envío de audio.
Comentarios	La decisión de reiniciar una llamada en la parte Asistente implica un cambio en el estado de la llamada en la parte del usuario. Así pues si el mensaje se perdiera o no fuera posible transferirlo a su destino, el asistente podría mandar más mensajes para modificar el estado.

Caso de uso	**13- Ver escritorio remoto.**
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	El asistente decide visualizar el escritorio del kiosco en vez de la cámara.

Precondiciones	Se ha realizado el caso de uso 07.
Postcondiciones	El asistente debe visualizar el escritorio remoto.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El asistente pulsa sobre la opción de visualizar el escritorio de la parte Kiosco.	2. La aplicación envía un mensaje indicando a la parte Kiosco que se desea visualizar el escritorio.
Extensiones síncronas	
-	-
Extensiones asíncronas	
-	1. En 2 no se consigue contactar con el servidor para el envío del mensaje, no se visualizará el escritorio (si no es el dispositivo ya seleccionado).
Comentarios	En caso de que el mensaje se perdiese y no se visualizara el dispositivo seleccionado, si el asistente pulsa sobre la opción de visualizar el escritorio remoto se volverá a enviar el mensaje.

Caso de uso	**14- Ver cámara remota.**
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	El asistente decide visualizar las imágenes provenientes de la cámara del Kiosco desde el que se realiza la llamada.
Precondiciones	Se ha realizado el caso de uso 07.
Postcondiciones	Se visualizan las imágenes capturadas por la cámara del kiosco.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El asistente pulsa sobre la opción de visualizar a cámara de la parte Kiosco.	2. Se envía un mensaje indicando a la parte Kiosco que se desea visualizar la cámara.
Extensiones síncronas	
	1. En 2 no se consigue contactar con el servidor para el envío del mensaje, no se visualizará la cámara (si no es el dispositivo ya seleccionado.
Extensiones asíncronas	
-	-
Comentarios	En caso de que el mensaje se perdiese y no se

	visualizara el dispositivo seleccionado, si el asistente pulsa sobre la opción de visualizar la cámara remota se volverá a enviar el mensaje.

Caso de uso	15- Iniciar.
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	El asistente inicia la parte Asistente y el sistema carga los datos necesarios del fichero de configuración.
Precondiciones	-
Postcondiciones	-
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	Sistema
1. El asistente inicia la aplicación.	2. El sistema lee todos los datos referentes a conexiones con servidores (SIP y MySQL), los usuarios SIP relacionados con el asistente y datos referentes a la transmisión de contenido multimedia.
	3. El sistema comprueba la estructura correcta de los datos. Si los datos son correctos se realiza el caso de uso 08.
Extensiones síncronas	
-	1. En 2 si no se encuentra el fichero se notifica al usuario y finaliza el sistema.
-	2. En 3 si los datos no poseen una estructura correcta el sistema lo notifica al asistente y finaliza.
Extensiones asíncronas	
-	-
Comentarios	-

Caso de uso	16- Notificar llamada.
Actor	AplicaciónKiosco.
Entorno de ejecución	Parte Asistente.
Resumen	Cuando se recibe una petición de llamada desde un kiosco el sistema debe de notificarlo al asistente.
Precondiciones	Se ha realizado los casos de uso 08 y 01.
Postcondiciones	-
Incluye	-

Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. AplicacionKiosco manda un mensaje notificando su petición de llamada.	2. Se recibe el mensaje de petición de llamada y se guardan los datos importantes de esta.
	3. La aplicación notifica al usuario de manera sonora y visual la recepción de una llamada.
Extensiones síncronas	
-	-
Extensiones asíncronas	
-	-
Comentarios	El mensaje enviado es el generado en el caso de uso 01.

Caso de uso	**17-Cancelación llamada.**
Actor	Asistente.
Entorno de ejecución	Parte Asistente.
Resumen	Mientras una llamada se está notificando al asistente (no se ha realizado ninguna acción sobre desde que se recibió) el usuario del Kiosco decide cancelarla.
Precondiciones	Se ha realizado el caso de uso 16 y el 05.
Postcondiciones	-
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. AplicacionKiosco envía el mensaje notificando la cancelación de la llamada.	2. Se recibe el mensaje de cancelación de la llamada.
	3. Se cesa en la notificación de la llamada.
	4. La parte cliente elimina toda información referente a dicha llamada.
Extensiones síncronas	
-	-
Extensiones asíncronas	
-	-
Comentarios	El mensaje enviado es el generado en el caso de uso 05.

Parte kiosco

Caso de uso	01- Llamar.
Actor	Usuario.
Entorno de ejecución	Parte Kiosco.
Resumen	Un usuario decide ponerse en contacto con un asistente.
Precondiciones	-
Postcondiciones	-
Incluye	02 y 03.
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	Sistema
1. El Usuario pulsa sobre el botón de asistencia remota.	2. Seguidamente se realiza el caso de uso 03 y posteriormente el 02.
	3. Se envía un mensaje al servidor indicando la petición llamada.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. En 3 si no se puede contactar con el servidor el sistema se lo notifica al usuario y finaliza la aplicación.
	2. En 3 si el servidor responde que el asistente no está disponible el sistema se lo notifica al usuario y finaliza la aplicación.
Comentarios	-

Caso de uso	02- Registro usuario.
Actor	Usuario.
Entorno de ejecución	Parte Kiosco.
Resumen	El sistema debe registrar el usuario en el servidor SIP a fin de identificar el Kiosco llamante y permitir realizar llamadas.
Precondiciones	Se ha realizado el caso de uso 03 y el caso 01 está en proceso.
Postcondiciones	-
Incluye	-
Extiende	01.
Hereda de	-
Flujo de Eventos	
Usuario	Sistema
	1. Envía los mensajes necesarios para el registro SIP en el servidor.

	2. El sistema responde aceptando el registro del usuario emparejado con el kiosco.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. En 1 no se puede establecer conexión con el servidor, el sistema lo comunica al cliente y finaliza.
	2. En 2 el servidor responde que el usuario no ha sido registrado con éxito, por tanto el registro falla, se le notifica al usuario y finaliza la aplicación.
Comentarios	-

Caso de uso	**03- Lectura de datos.**
Actor	Usuario.
Entorno de ejecución	Parte Kiosco.
Resumen	El sistema lee los datos necesarios para la comunicación con el servidor SIP y posiblemente para el almacenamiento de datos importantes de las llamadas. Se leen desde el archivo de configuración.
Precondiciones	El caso de uso 01 está en proceso.
Postcondiciones	Se cargan todos los datos requeridos por el sistema.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
	1. Se leen los datos requeridos por la aplicación desde el fichero de configuración.
	2. Se comprueba la estructura de los datos.
Extensiones síncronas	
	1. En 1 si en el fichero de configuración se indica que la lectura de datos se hará desde una base de datos se realizará el caso de uso 04.
	2. En 2 si los datos no cumplen las estructuras correctas, se le notifica al usuario y finaliza el sistema.
Extensiones asíncronas	
-	-
Comentarios	-

Caso de uso	**04- Lectura de datos desde servidor.**
Actor	Usuario.

Entorno de ejecución	Parte Kiosco.
Resumen	El sistema lee los datos necesarios para la comunicación con el servidor SIP y posiblemente para el almacenamiento de datos importantes de las llamadas. Los datos se leen de una BD.
Precondiciones	El caso de uso 01 está en proceso y en el archivo de configuración se indica que los datos deben ser leídos desde BD.
Postcondiciones	Se cargan todos los datos requeridos por el sistema.
Incluye	-
Extiende	-
Hereda de	03.
Flujo de Eventos	
Usuario	**Sistema**
	1. Se leen los datos del servidor MySQL fuente desde el archivo de configuración.
	2. Se realiza la consulta pertinente al servidor.
	3. Se cargan los datos y se actualiza el fichero de configuración.
Extensiones síncronas	
	1. En 1 los datos de conexión no tienen una estructura correcta, se le notifica al usuario y finaliza el sistema.
	2. En 2 la conexión con el servidor no se realiza de forma satisfactoria, se le notifica al usuario y finaliza el sistema.
	3. En 2 si los datos no cumplen las estructuras correctas, se le notifica al usuario y finaliza el sistema.
Extensiones asíncronas	
-	-
Comentarios	-

Caso de uso	**05- Cancelar llamada.**
Actor	Usuario.
Entorno de ejecución	Parte Kiosco.
Resumen	Después de realizar una petición de llamada el usuario decide cancelar la petición de comunicación.
Precondiciones	Se ha realizado el caso de uso 01 y aún no se ha realizado 07.
Postcondiciones	No se establece comunicación con el asistente.
Incluye	-
Extiende	-

Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El usuario decide no establecer comunicación con el asistente.	2. Se envía al servidor la petición de cancelación de llamada y el sistema finaliza.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. En 2 la conexión con el servidor no se realiza de forma satisfactoria, se notifica el error al usuario y finaliza el sistema.
Comentarios	-

Caso de uso	**18- Empezar llamada.**
Actor	AplicacionAsistente.
Entorno de ejecución	Parte Kiosco.
Resumen	El asistente acepta una llamada y la parte Kiosco realiza las acciones necesarias para la comunicación.
Precondiciones	Se ha realizado el caso de uso 07.
Postcondiciones	Se abre una ventana donde se observan las imágenes recibidas y enviadas. También se empieza a reproducir el audio recibido y a enviar el capturado.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. AplicaciónAsistente envía un mensaje notificando la aceptación de la llamada.	2. La parte Kiosco recibe el mensaje y notifica la recepción de la aceptación mediante un mensaje.
	3. Se guardan los datos importantes para la transmisión y recepción de audio y video a partir del mensaje recibido.
	4. Se inicia el envío y recepción de audio y video.
Extensiones síncronas	
-	-
Extensiones asíncronas	
-	-
Comentarios	El mensaje enviado es el generado en el caso de uso 07.

Caso de uso	**19- Tratar rechazo llamada.**
Actor	AplicacionAsistente.
Entorno de ejecución	Parte Kiosco.

Resumen	El asistente ha decidido rechazar una llamada, el kiosco notifica el rechazo.
Precondiciones	Se ha realizado el caso de uso 06.
Postcondiciones	-
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. AplicacionAsistente envía un mensaje notificando el rechazo de la llamada.	2. La parte Kiosco recibe el mensaje informando del rechazo de la llamada.
	3. Se le notifica al usuario la decisión del asistente y finaliza el sistema.
Extensiones síncronas	
-	-
Extensiones asíncronas	
-	-
Comentarios	El mensaje enviado es el generado en el caso de uso 06.

Caso de uso	**20- Esperar.**
Actor	AplicacionAsistente.
Entorno de ejecución	Parte Kiosco.
Resumen	El asistente decide poner a la espera un kiosco, dicho kiosco tan solo iniciar la comunicación de video.
Precondiciones	Se ha realizado el caso de uso 09 y 18.
Postcondiciones	El usuario es incapaz de comunicarse con el asistente.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. AplicacionAsistente envía un mensaje notificando la espera de la llamada.	2. El sistema recibe el mensaje y confirma la recepción mediante otro mensaje.
	3. Se recibe la aceptación de la llamada y se responde con un ACK.
	3. El sistema tan solo inicia el envío de video.
Extensiones síncronas	
-	-
Extensiones asíncronas	
	1. En 2 no se recibe la aceptación de la llamada, el caso de uso termina y no se inicia la comunicación.
Comentarios	El mensaje enviado es el generado en el caso de uso 09.

Caso de uso	21- Mostrar escritorio.
Actor	AplicacionAsistente.
Entorno de ejecución	Parte Kiosco.
Resumen	El asistente decide visualizar el escritorio del Kiosco, por tanto el sistema debe cambiar (si es necesario) la fuente de captura de imágenes.
recondiciones	Se ha realizado el caso de uso 18.
Postcondiciones	Se envían las capturas de pantalla realizadas sobre el escritorio.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	Sistema
1. AplicacionAsistente envía un mensaje informando de la selección del escritorio como fuente de las imágenes.	2. El sistema recibe el mensaje y confirma la recepción de este mediante otro mensaje.
	3. Se cambia la fuente de captura de imágenes.
Extensiones síncronas	
-	-
Extensiones asíncronas	
-	-
Comentarios	El mensaje enviado es el generado en el caso de uso 13.

Caso de uso	22- Activar visón cámara.
Actor	AplicacionAsistente.
Entorno de ejecución	Parte Kiosco.
Resumen	El asistente decide visualizar la camara del Kiosco, por tanto el sistema debe cambiar (si es necesario) la fuente de captura de imágenes.
Precondiciones	Se ha realizado el caso de uso 18.
Postcondiciones	Se envían las imágenes procedentes de la cámara.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	Sistema
1. AplicacionAsistente envía un mensaje informando de la selección de la cámara como fuente de las imágenes.	2. El sistema recibe el mensaje y confirma la recepción de este mediante otro mensaje.
	3. Se cambia la fuente de captura de imágenes.
Extensiones síncronas	

-	-
Extensiones asíncronas	
-	-
Comentarios	El mensaje enviado es el generado en el caso de uso 14.

Caso de uso	**23- Activar llamada.**
Actor	AplicaciónAsistente.
Entorno de ejecución	Parte Kiosco.
Resumen	Cuando el asistente decide activar/reiniciar una llamada la parte Kiosco vuelve a activar el envío y recepción de audio y video.
Precondiciones	Se ha realizado el caso de uso 20 o 26.
Postcondiciones	El usuario puede volver a comunicarse con el asistente.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. AplicacionAsistente envía un mensaje informando de la activación de la llamada.	2. El sistema recibe el mensaje y confirma la recepción de este mediante otro mensaje.
	3. Se inicia el envío y recepción de audio y video.
Extensiones síncronas	
-	-
Extensiones asíncronas	
-	-
Comentarios	El mensaje enviado es el generado en el caso de uso 12.

Caso de uso	**24- Predeterminación dispositivo de audio.**
Actor	Técnico.
Entorno de ejecución	Parte Kiosco.
Resumen	La primera vez que se inicia la parte Kiosco el técnico tiene la capacidad de seleccionar el dispositivo de audio predeterminado para la captura de datos.
Precondiciones	Existen dispositivos de audio compatibles y aún no se ha seleccionado ninguno de estos.
Postcondiciones	-
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El técnico inicia la aplicación.	2. Selecciona el dispositivo de audio para la

	captura de datos.
3. El técnico confirma los cambios.	
Extensiones síncronas	
-	1. En 2 no existen dispositivos compatibles, se le notifica al técnico y finaliza el sistema.
Extensiones asíncronas	
-	-
Comentarios	-

Caso de uso	**25- Predeterminación dispositivo de video.**
Actor	Técnico.
Entorno de ejecución	Parte Kiosco.
Resumen	La primera vez que se inicia la parte Kiosco el técnico tiene la capacidad de seleccionar el dispositivo de video predeterminado para la captura de datos.
Precondiciones	Existen dispositivos de audio compatibles y aún no se ha seleccionado ninguno de estos.
Postcondiciones	-
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	**Sistema**
1. El técnico inicia la aplicación.	2. Selecciona el dispositivo de video para la captura de datos.
3. El técnico confirma los cambios.	
Extensiones síncronas	
-	1. En 2 no existen dispositivos compatibles, se le notifica al técnico y finaliza el sistema.
Extensiones asíncronas	
-	-
Comentarios	-

Caso de uso	**26- Pausar llamada asistente.**
Actor	AplicaciónAsistente.
Entorno de ejecución	Parte Kiosco.
Resumen	El asistente decide pausar una llamda ya empezada y el Kiosco tan solo envía imágenes del dispositivo seleccionado.
Precondiciones	Se ha realizado el caso de uso 18 o 12.
Postcondiciones	El usuario es incapaz de comunicarse con el asistente.
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	

Usuario	Sistema
1. AplicacionAsistente envía un mensaje notificando la pausa de la llamada.	2. El sistema recibe el mensaje y confirma la recepción mediante otro mensaje.
	3. Se cesa en el envió y reproducción de audio. No obstante el video tan solo se deja de recibir.
Extensiones síncronas	
-	-
Extensiones asíncronas	
-	-
Comentarios	El mensaje enviado es el generado en el caso de uso 11. La recepción del mensaje indicando la pausa de la llamada puede recibirse varias veces (caso de mala conexión a internet en alguna de las dos partes), no obstante solo tendrá efecto el primer mensaje recibido.

Caso de uso	27- Finalizar llamada asistente.
Actor	AplicaciónAsistente.
Entorno de ejecución	Parte Kiosco.
Resumen	El asistente finaliza una llamada y la parte Kisoco finaliza.
Precondiciones	Se ha realizado el caso de uso 18.
Postcondiciones	-
Incluye	-
Extiende	-
Hereda de	-
Flujo de Eventos	
Usuario	Sistema
1. AplicacionAsistente envía un mensaje informando de la finalización de la llamada.	2. El sistema recibe el mensaje y confirma la recepción de este mediante otro mensaje.
	3. Termina el envío y recepción multimedia y finaliza el sistema.
Extensiones síncronas	
-	-
Extensiones asíncronas	
-	-
Comentarios	-

Anexo C

Diagramas secuencia

Parte asistente

- 06-Rechazar llamada. Identificador del diagrama: **sd RechazarLlamada**.

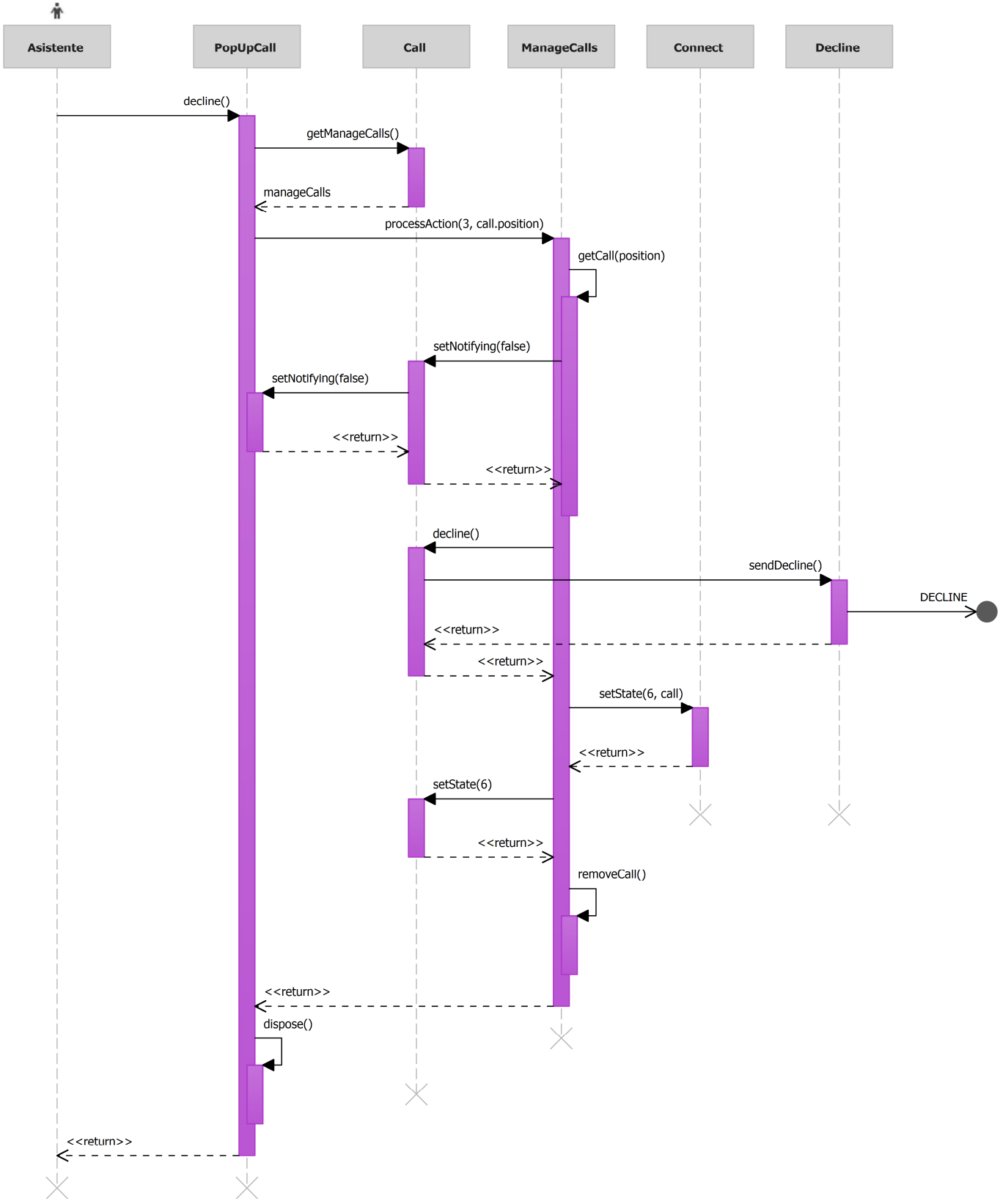


Ilustración 57: sd RechazarLlamada.

- 07-Aceptar llamada. Identificador del diagrama: **sd AceptarLlamada**.

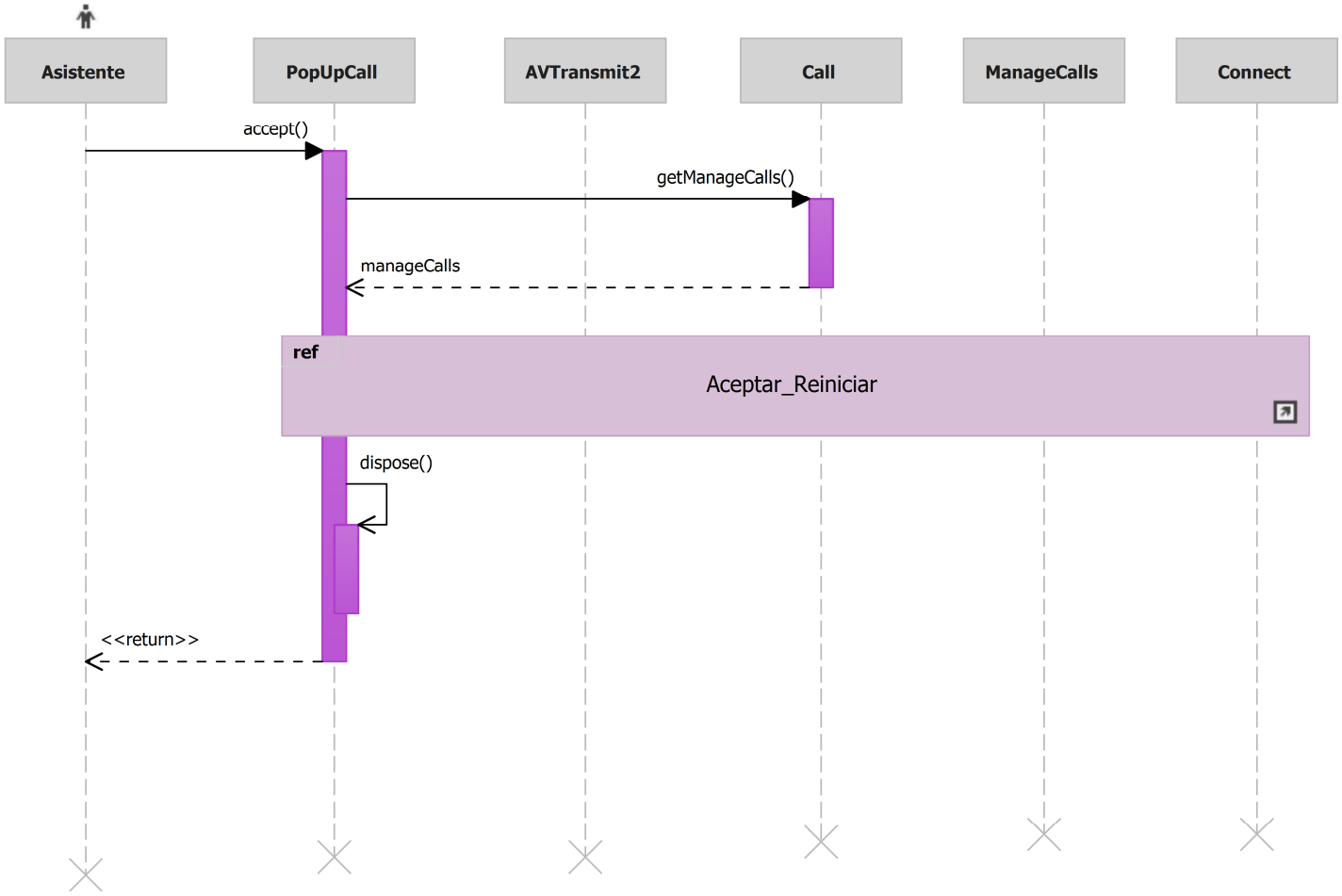


Ilustración 58: sd AceptarLlamada.

- 08-Registro asistente. Identificador del diagrama: **sd RegistroAsistente**.

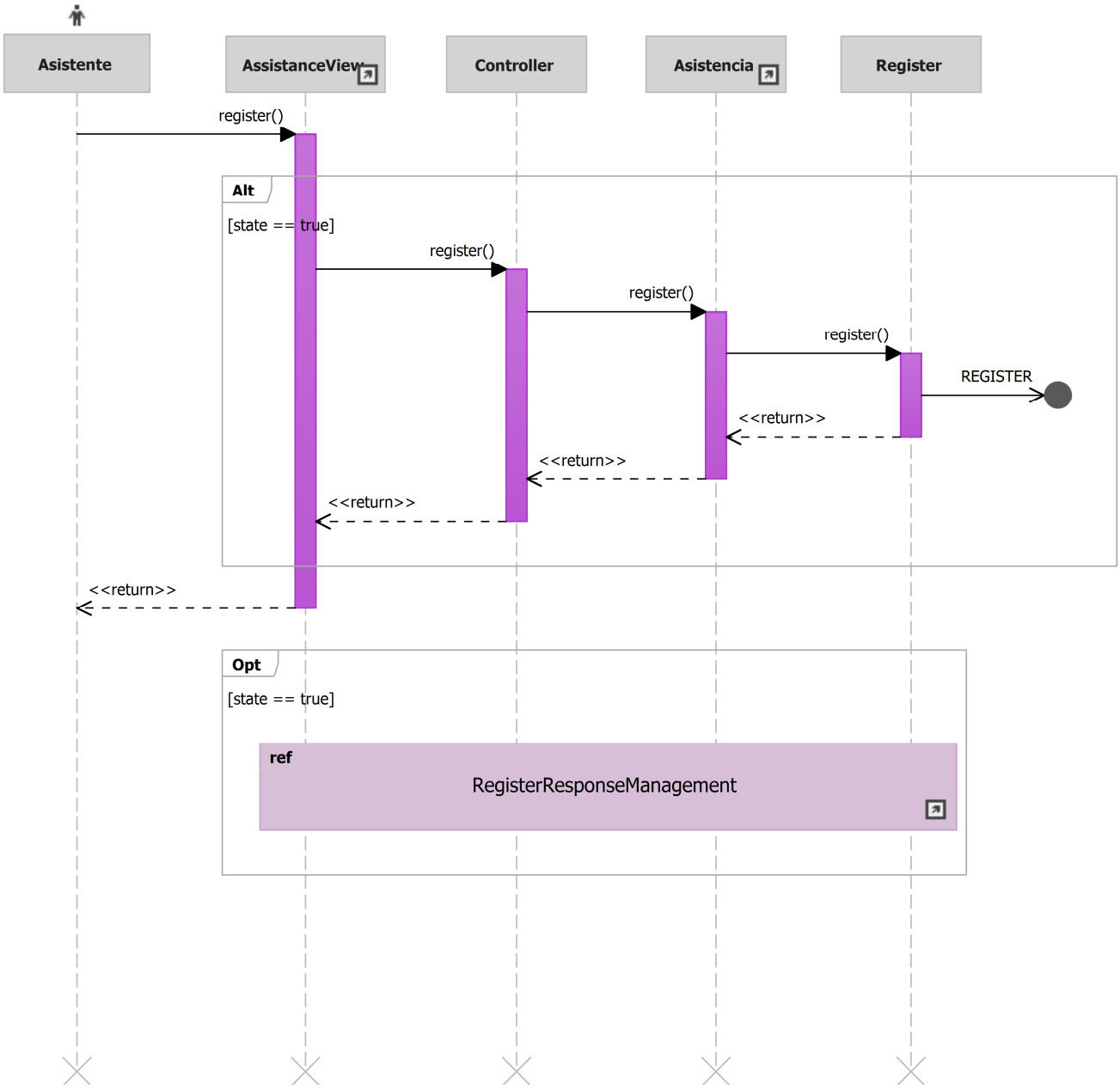


Ilustración 59: sd RegistroAsistente.

- 09-Poner a la espera. Identificador del diagrama: **sd PonerALaEspera**.

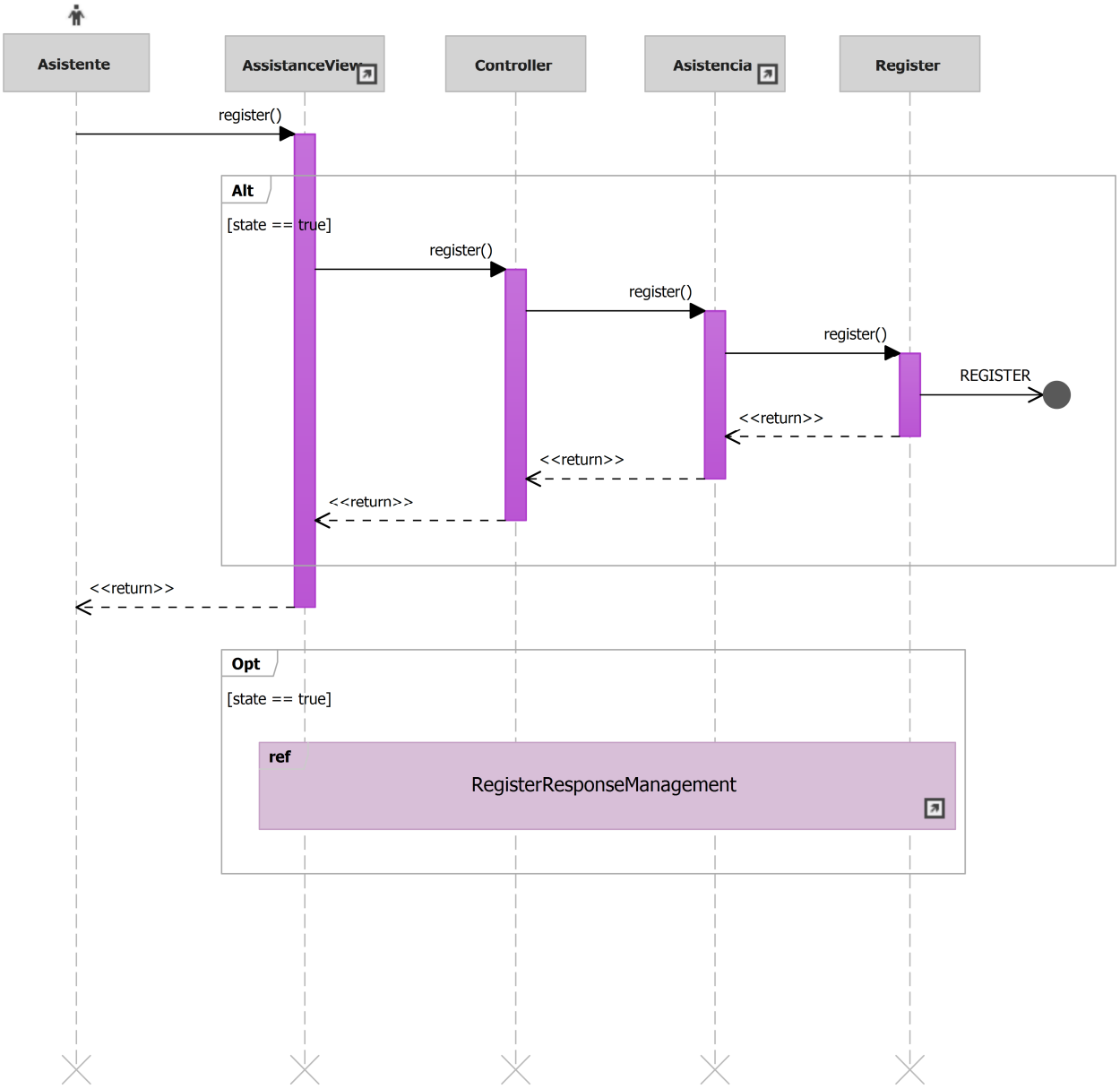


Ilustración 60: sd PonerALaEspera.

- 10-Finalizar llamada asistente. Identificador del diagrama: **sd FinalizarLlamadaAsistente**.

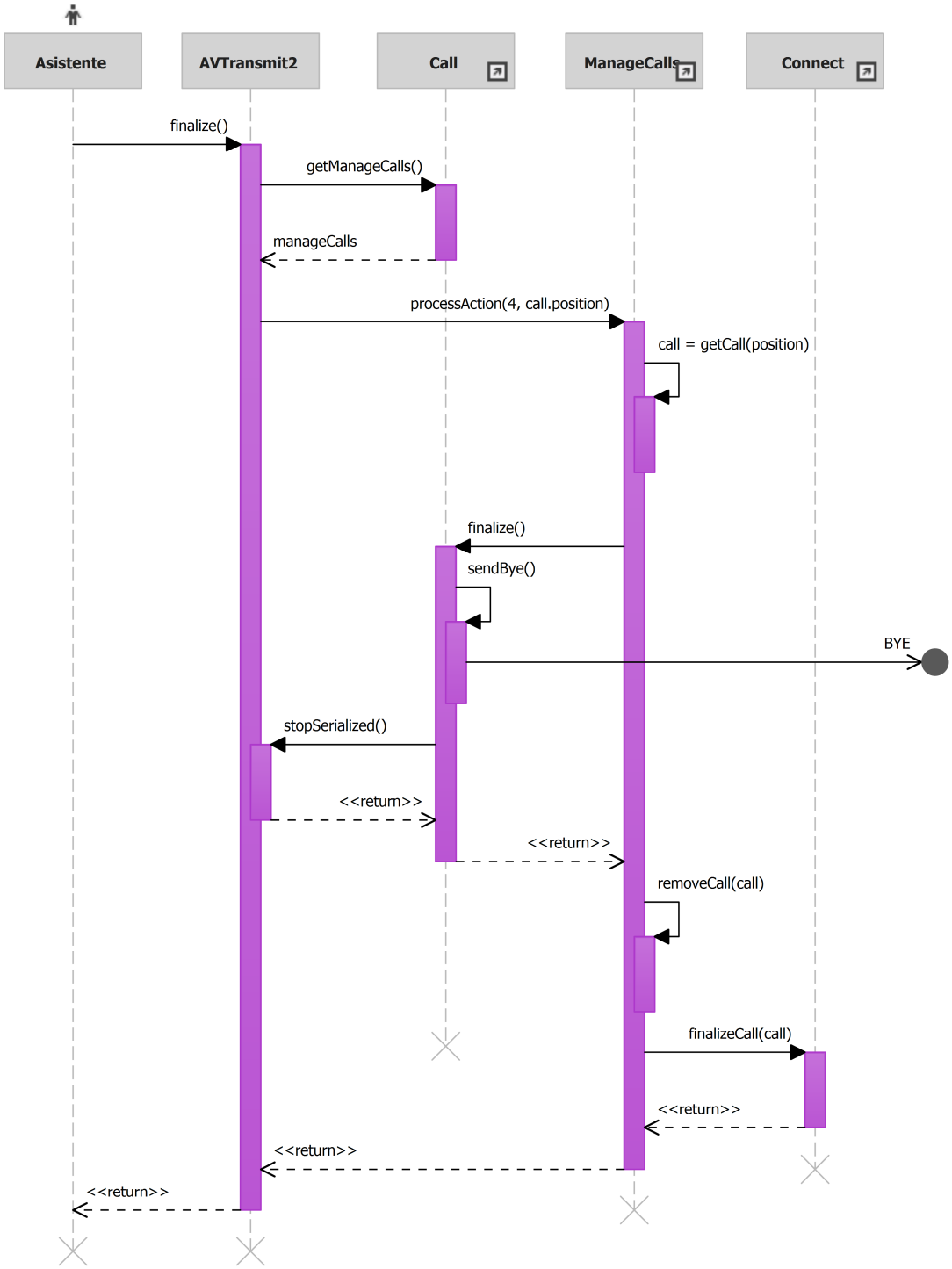


Ilustración 61: sd FinalizarLlamadaAsistente.

- 11-Pausar llamada asistente. Identificador del diagrama: **sd PausarLlamadaAsistente**.

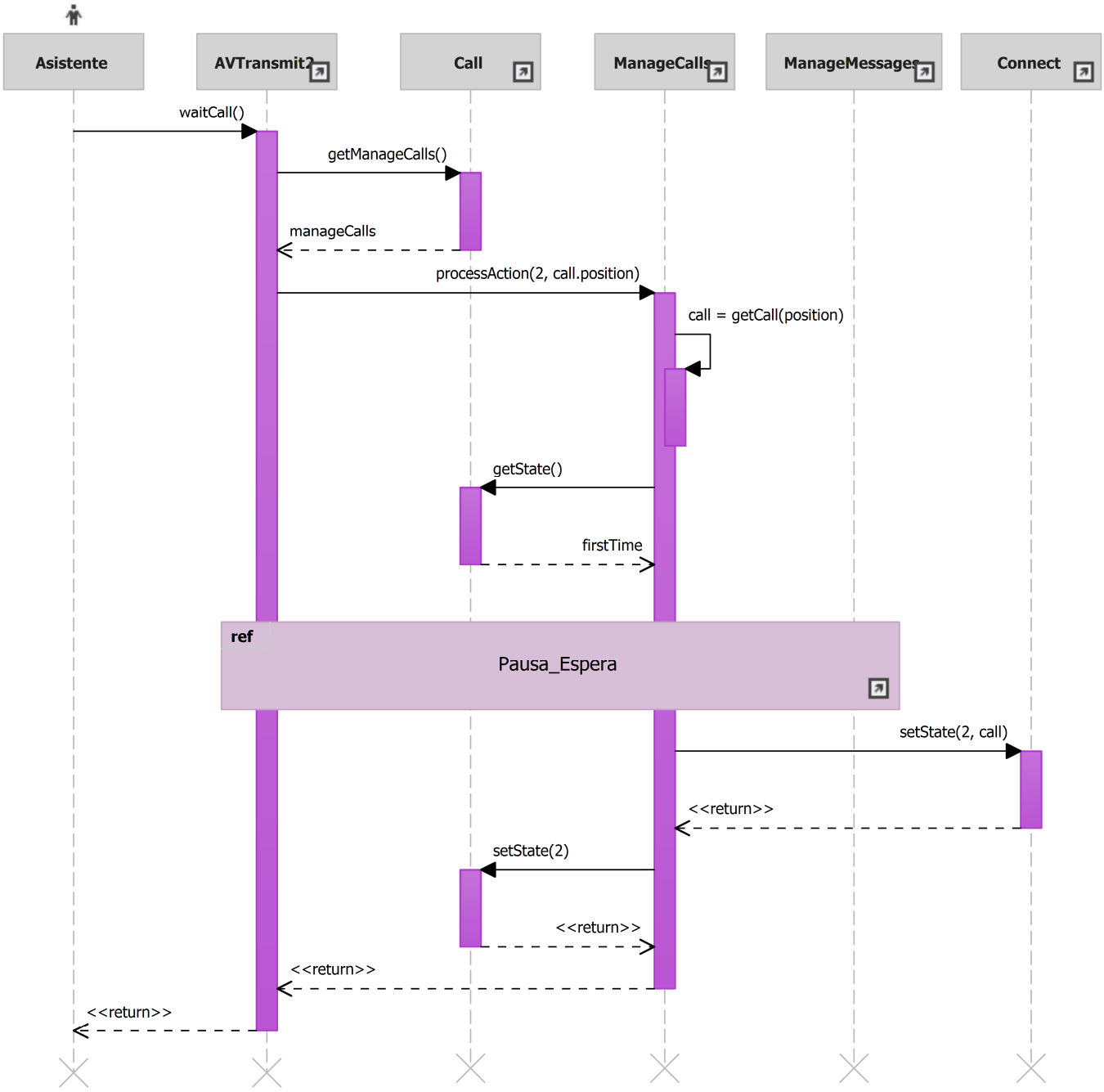


Ilustración 62: sd PausarLlamadaAsistente.

- 12-Reiniciar llamada. Identificador del diagrama: **sd ReiniciarLlamada**.

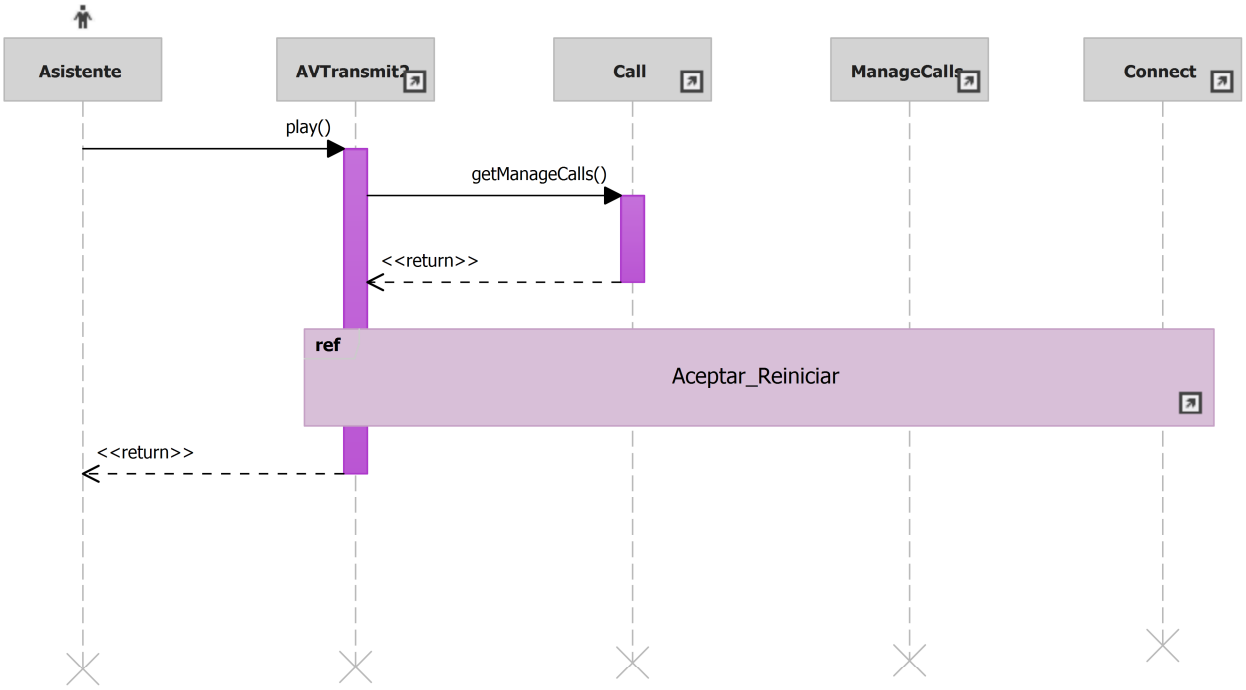


Ilustración 63: sd ReiniciarLlamada.

- 13-Ver escritorio remoto. Identificador del diagrama: **sd VerEscritorioRemoto**.

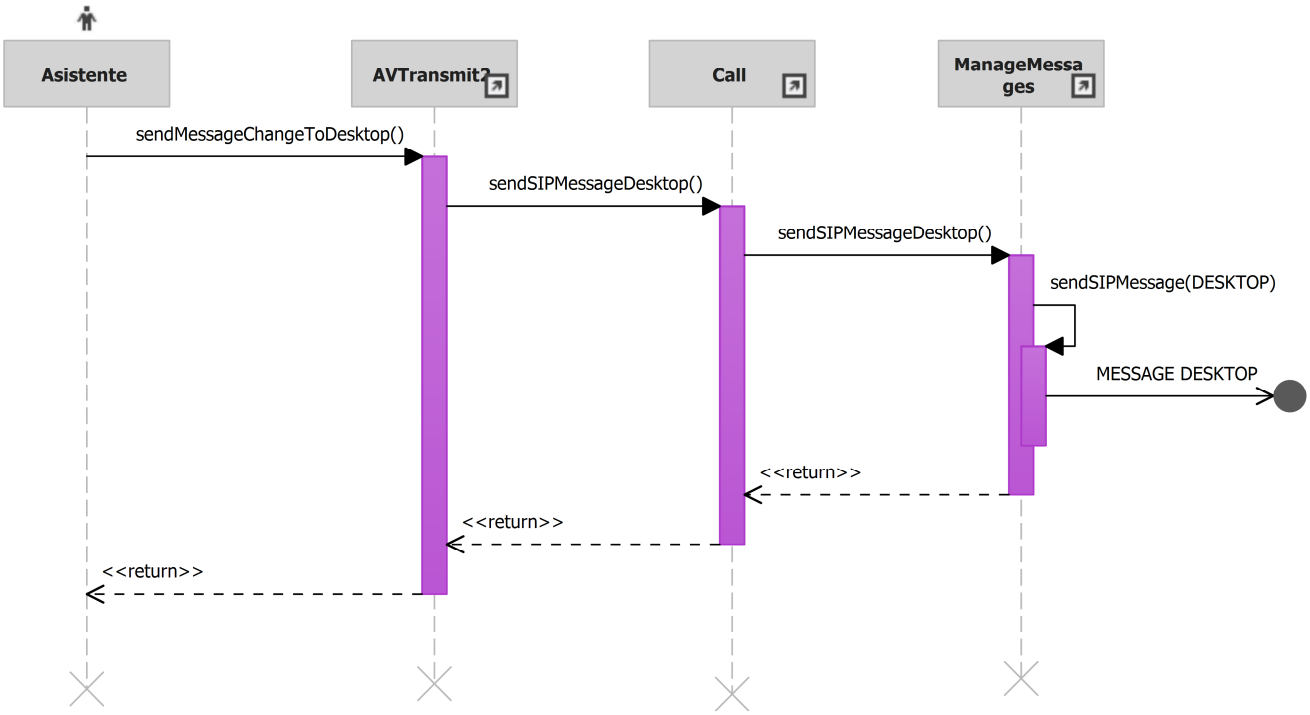


Ilustración 64: sd VerEscritorioRemoto.

- 14-Ver cámara remota. Identificador del diagrama: **sd VerCamaraRemota**.

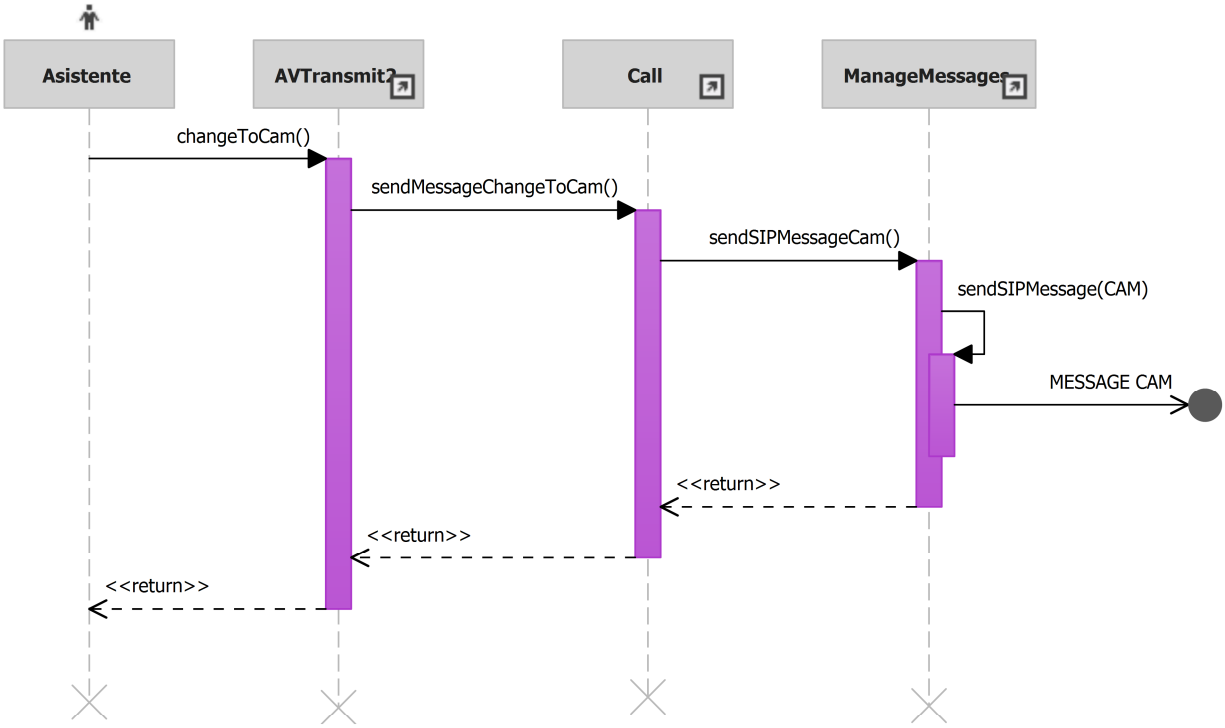


Ilustración 65: sd VerCamaraRemota.

- 15 – Iniciar. Identificador del diagrama: **sd Iniciar**.

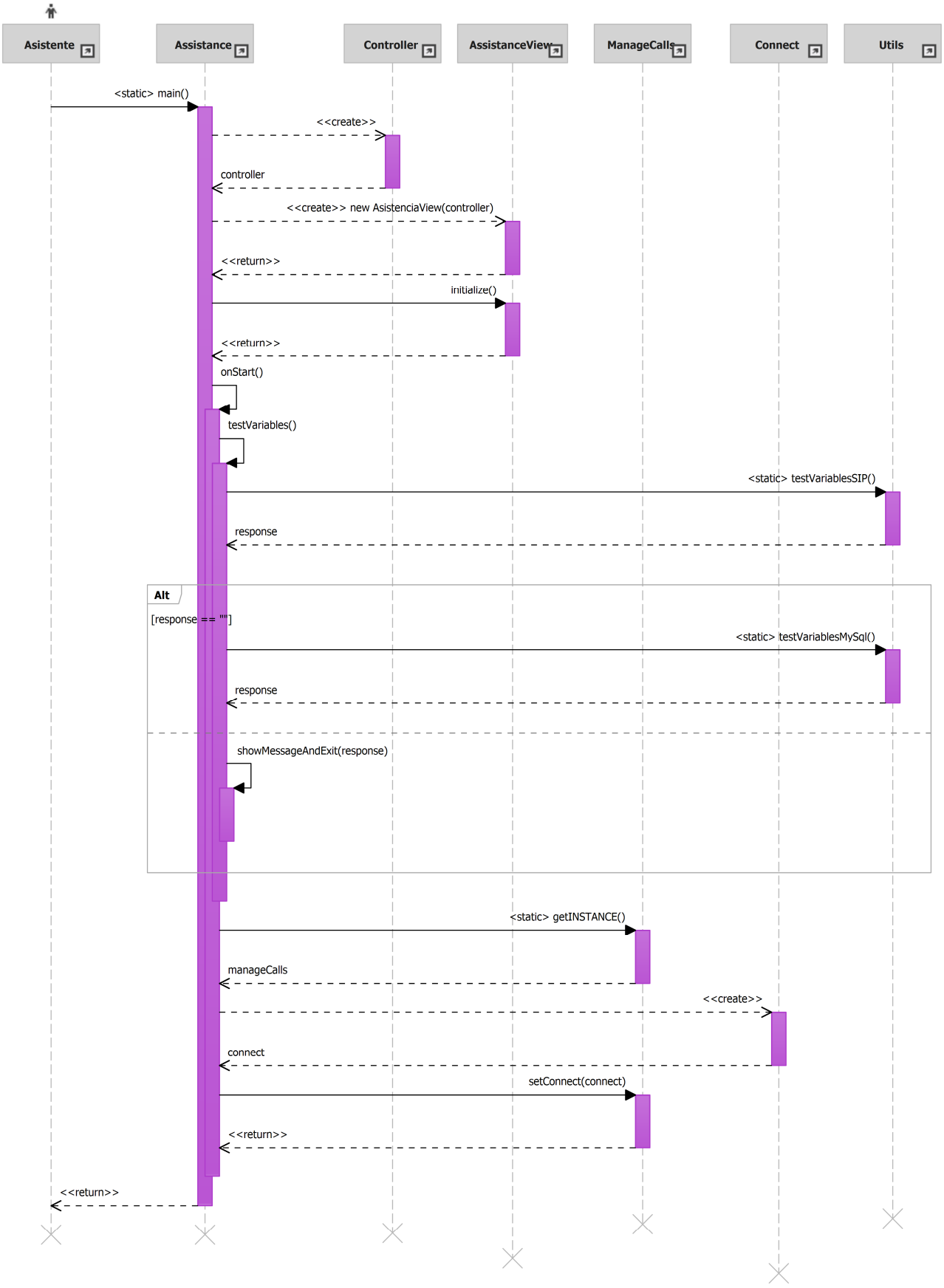
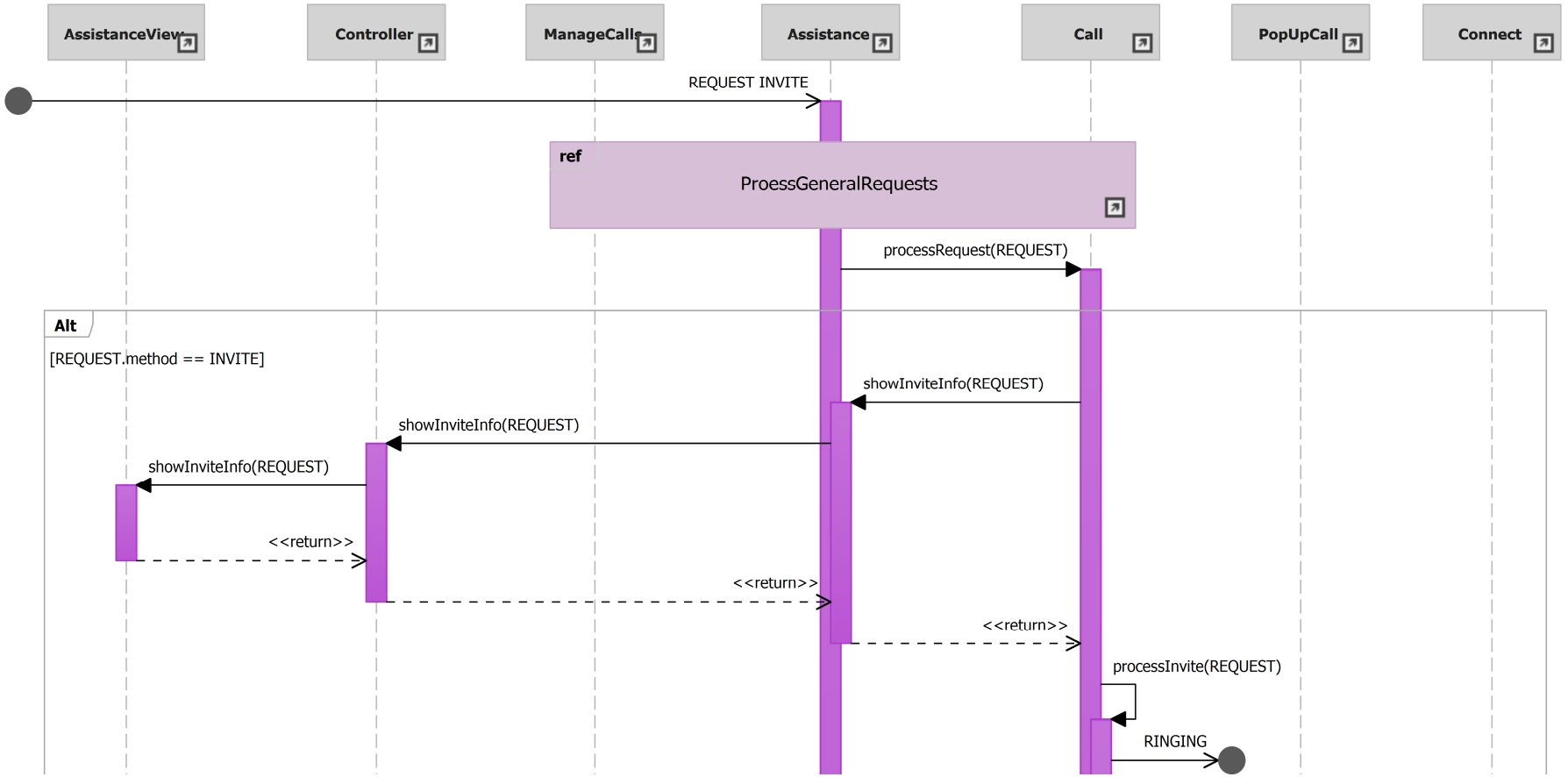


Ilustración 66: sd Iniciar.

- 16-Notificar llamada. Identificador del diagrama: **sd NotificarLlamada**.



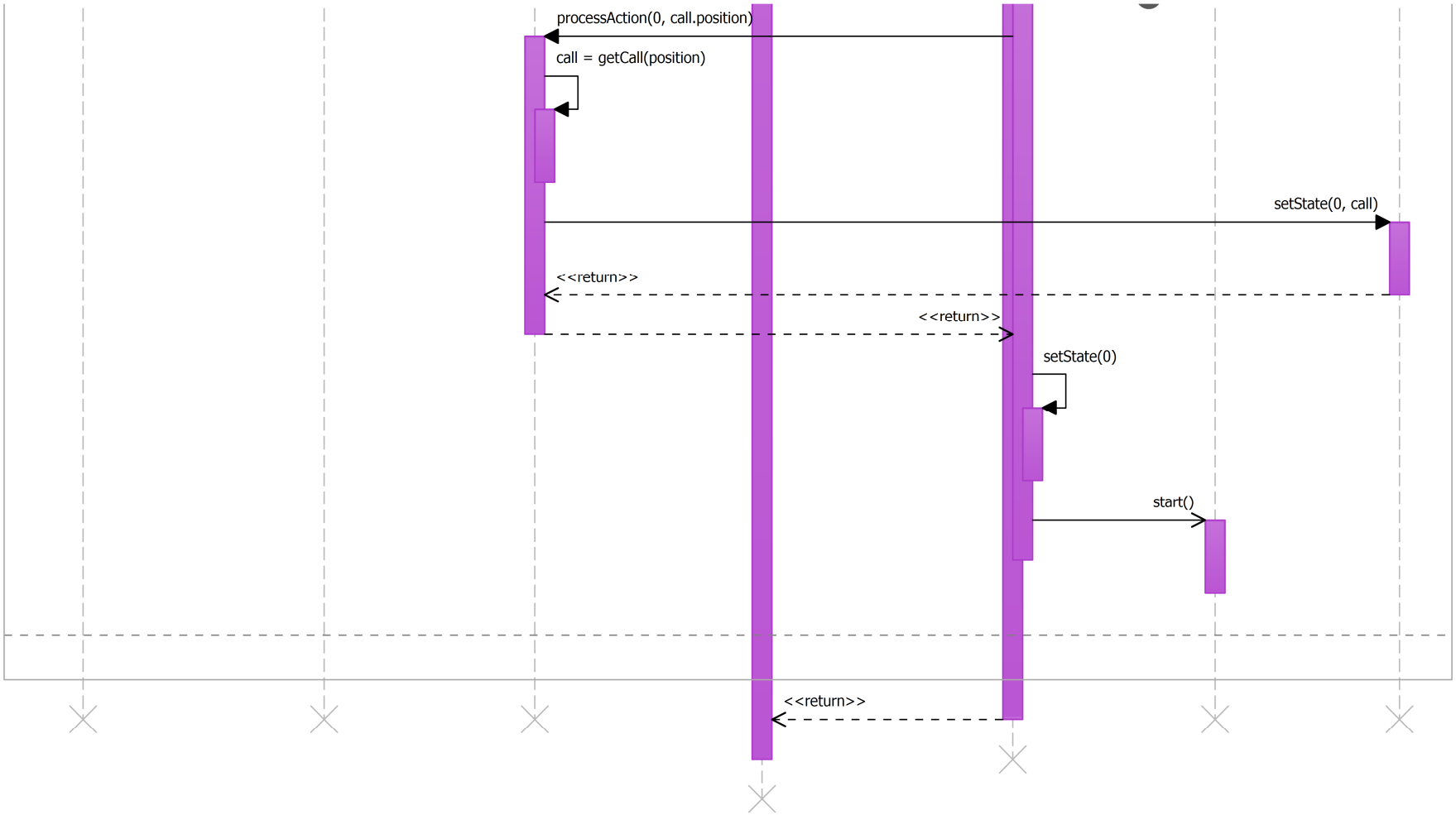


Ilustración 67: sd NotificarLlamada.

- 17-Cancelacion llamada. Identificador del diagrama: **sd CancelacionLlamada**.

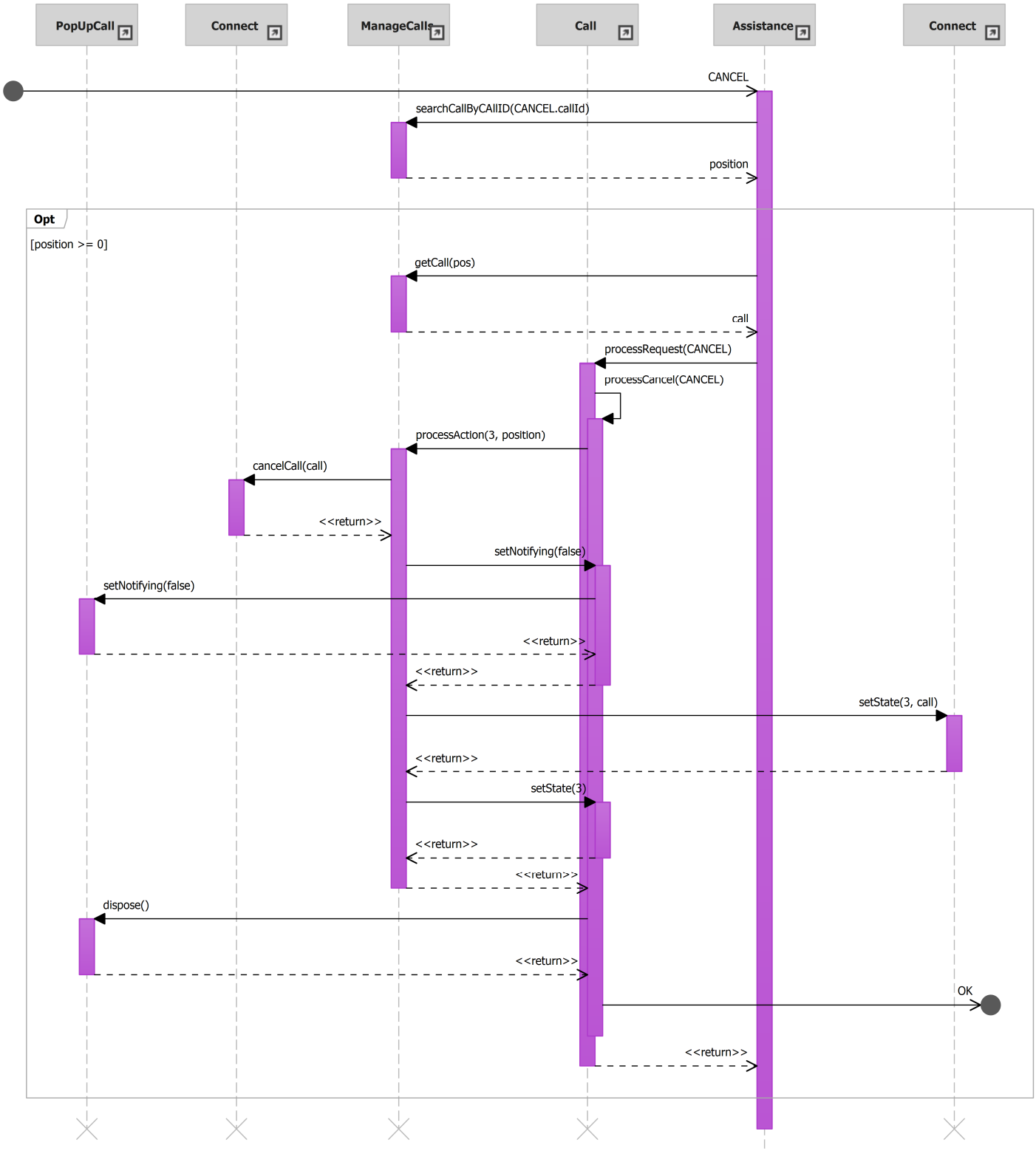
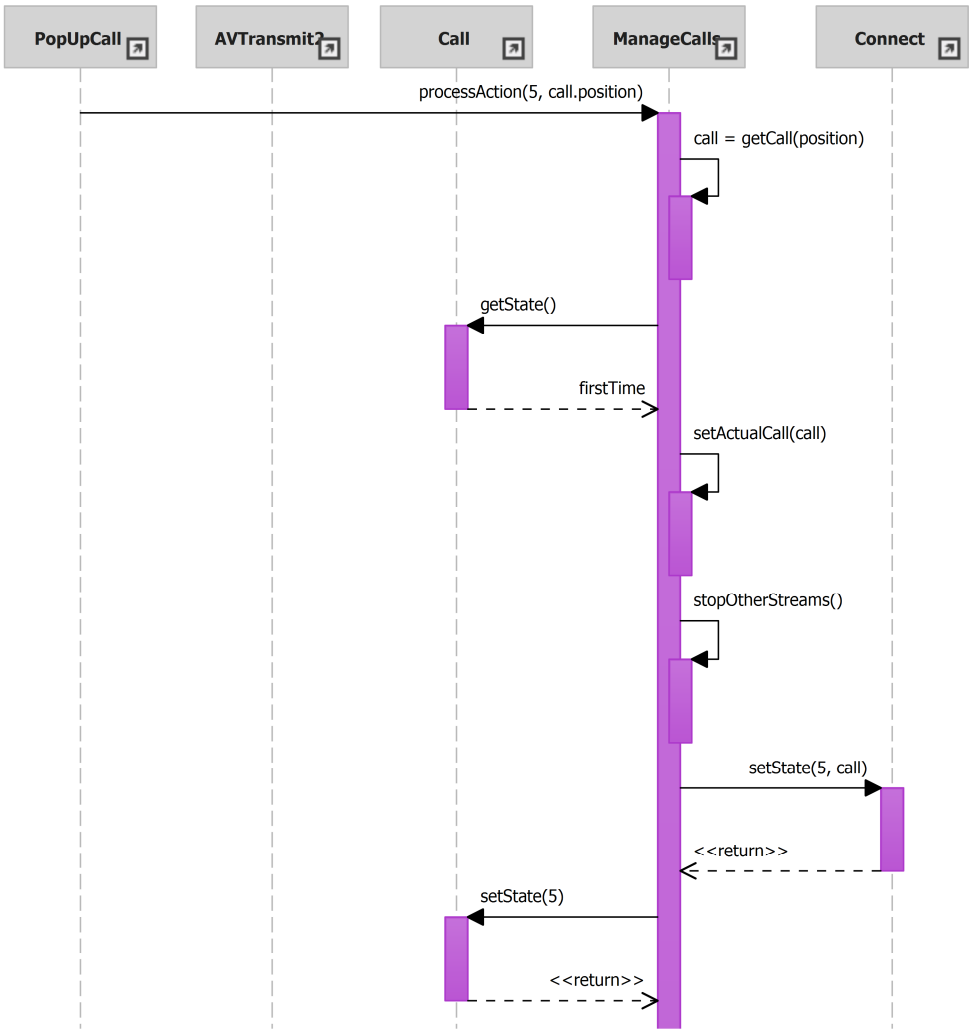


Ilustración 68: sd CancelacionLlamada.

Diagramas auxiliares

- sd Aceptar_Reiniciar.



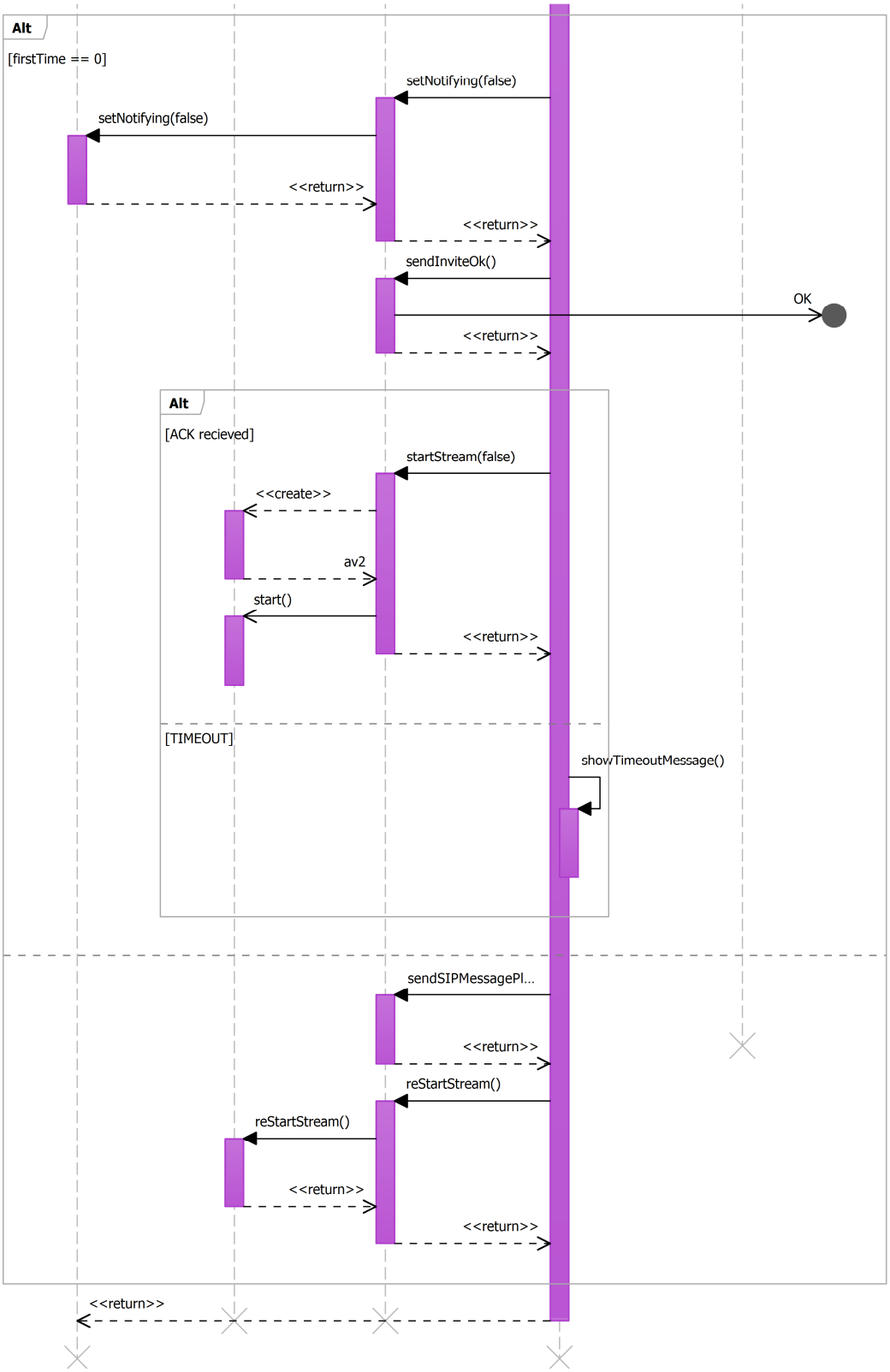


Ilustración 69: sd Aceptar_Reiniciar.

- sd RegisterResponseManagement.

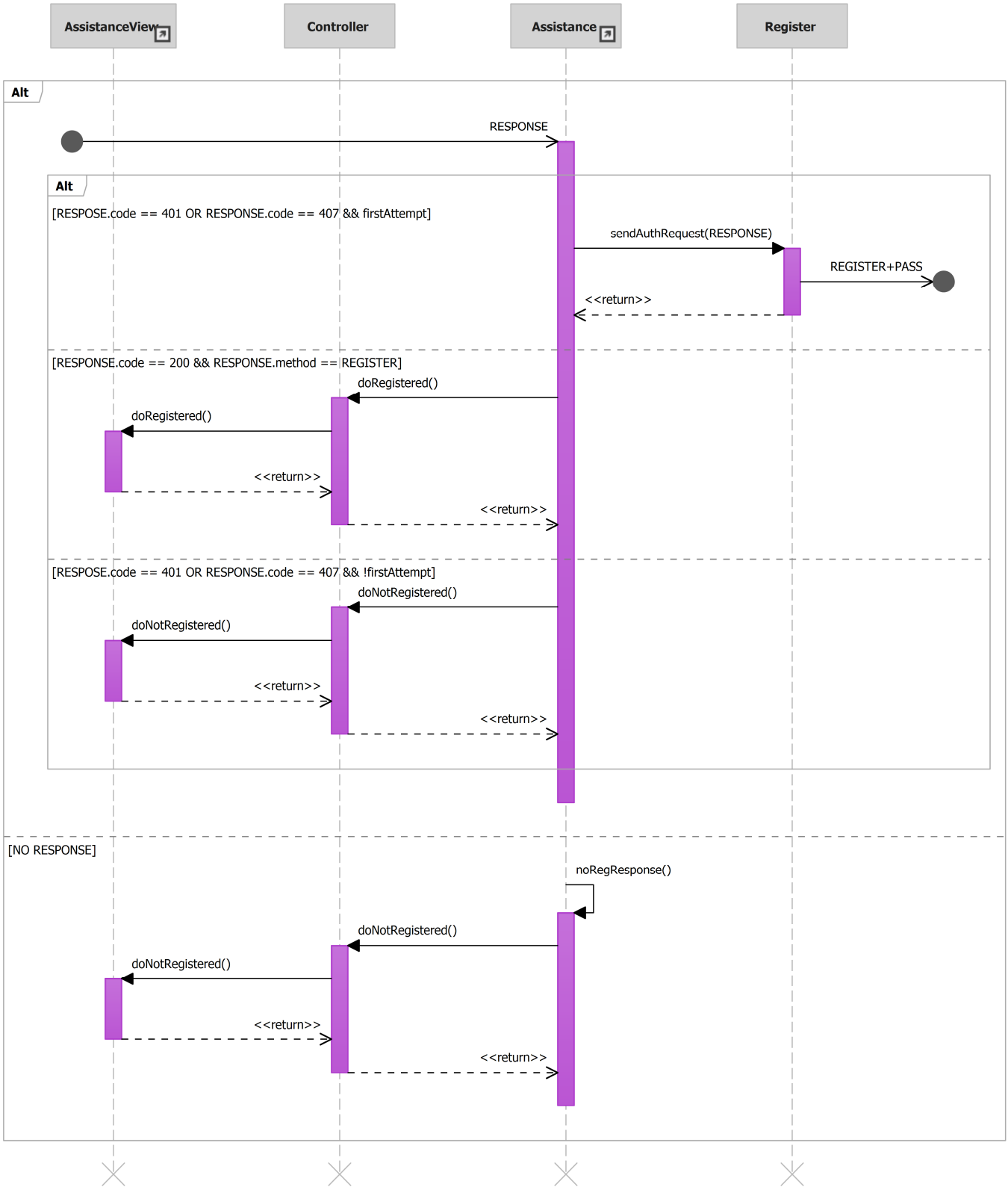


Ilustración 70: sd RegisterResponseManagement.

- sd Pausa_Espera.

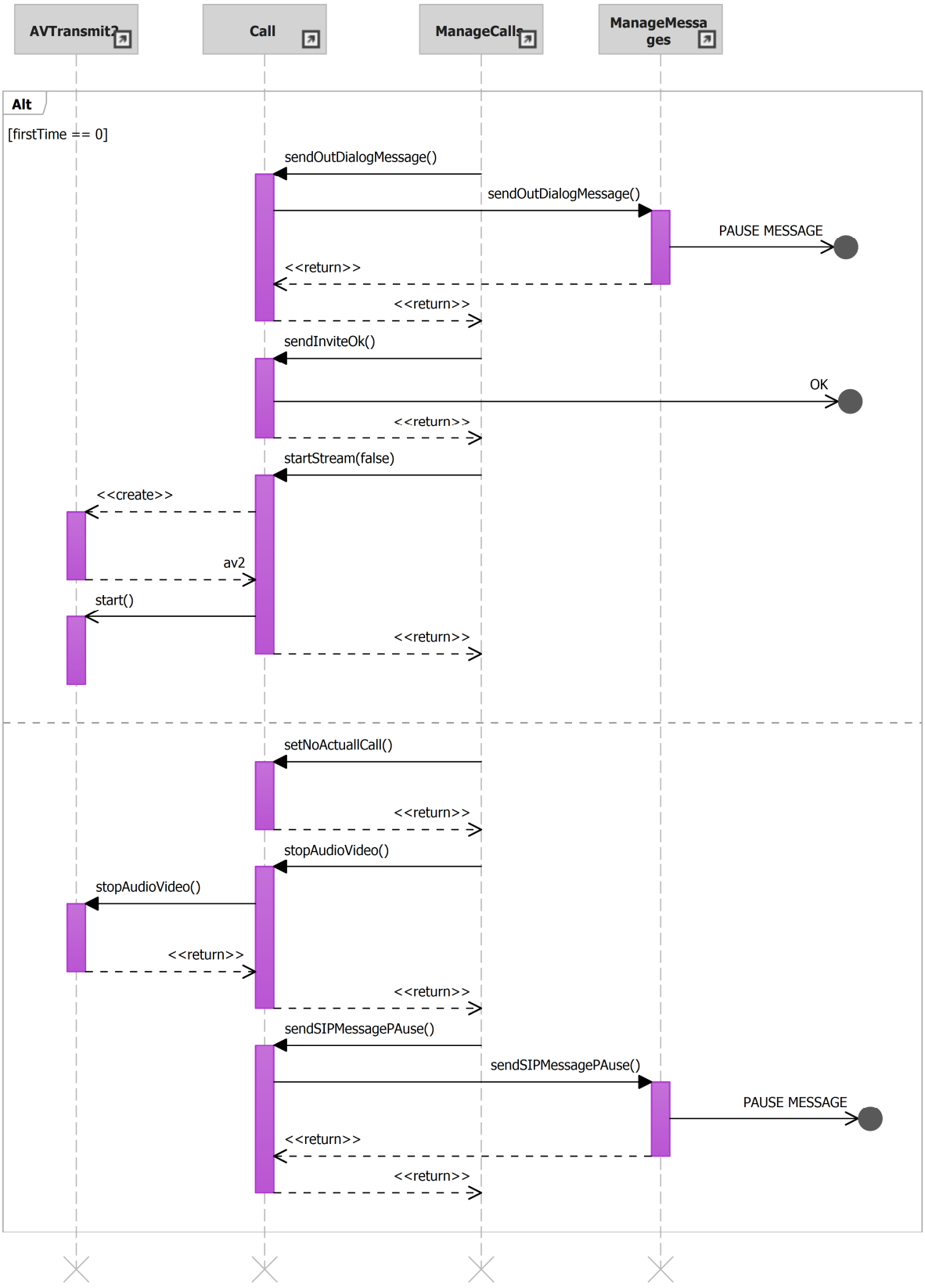


Ilustración 71: sd Pausa_Espera.

- sd ProessGeneralRequests.

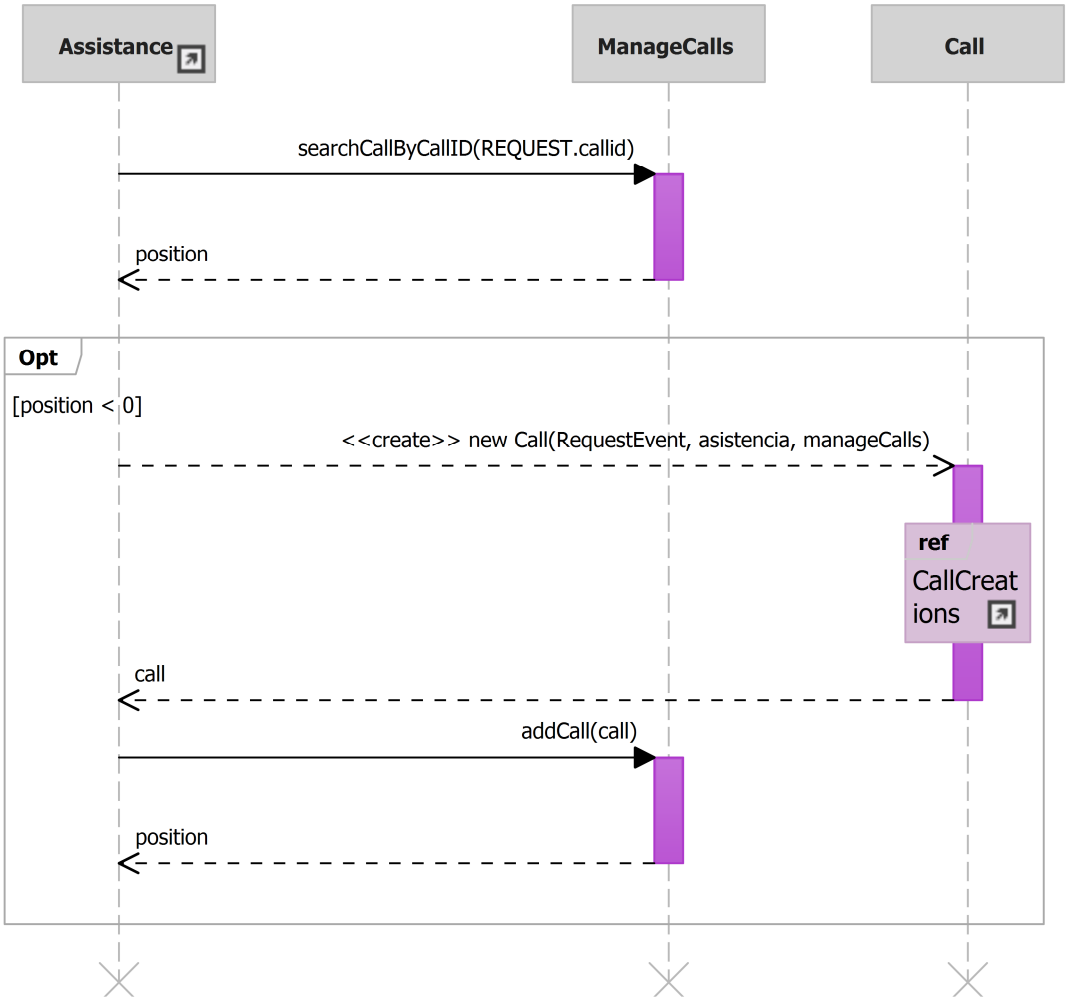


Ilustración 72: sd ProessGeneralRequests.

- sd CallCreations.

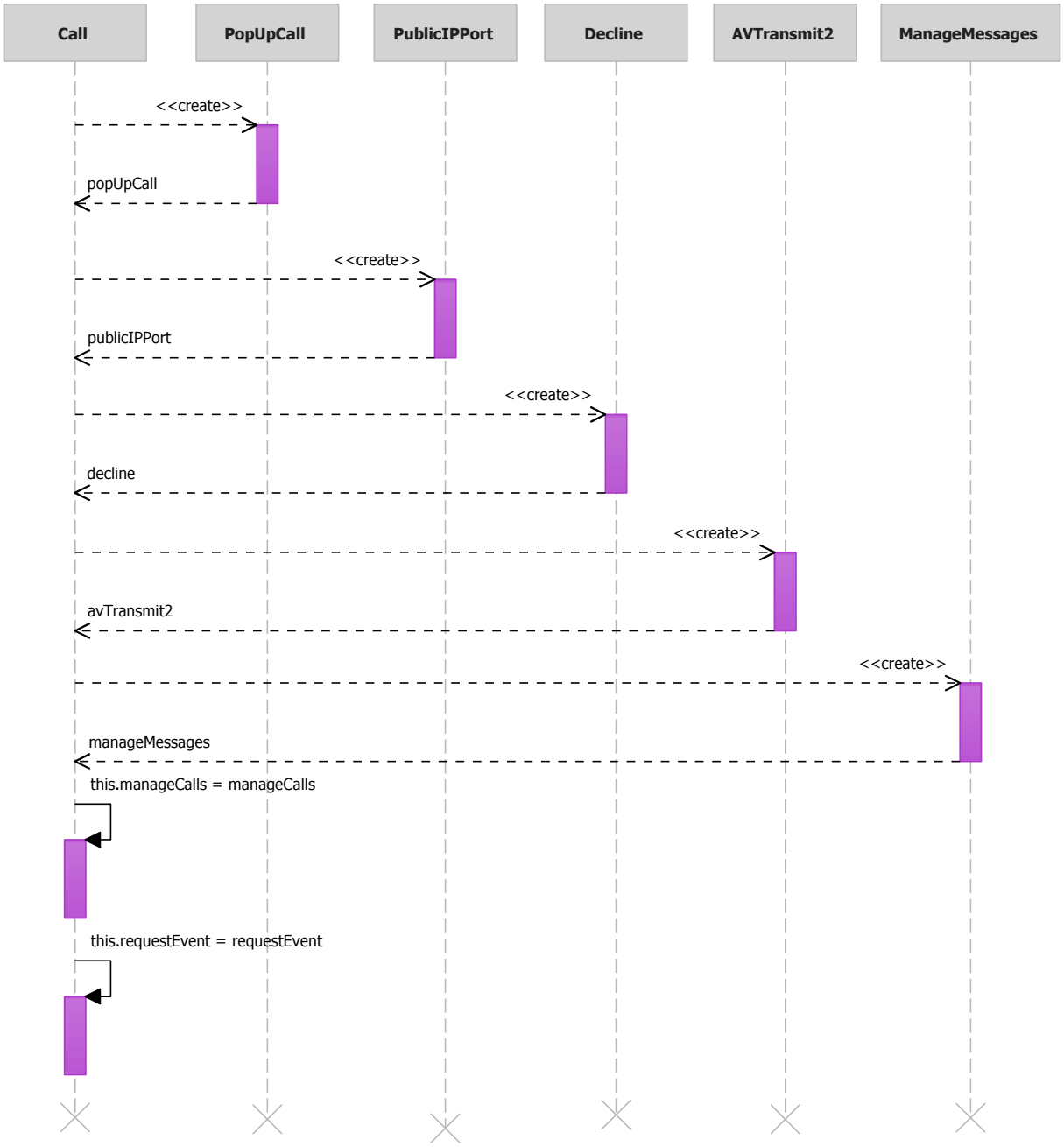


Ilustración 73: sd CallCreations.

Parte kiosco

- 01- Llamar. Identificador del diagrama: **sd Llamar**.

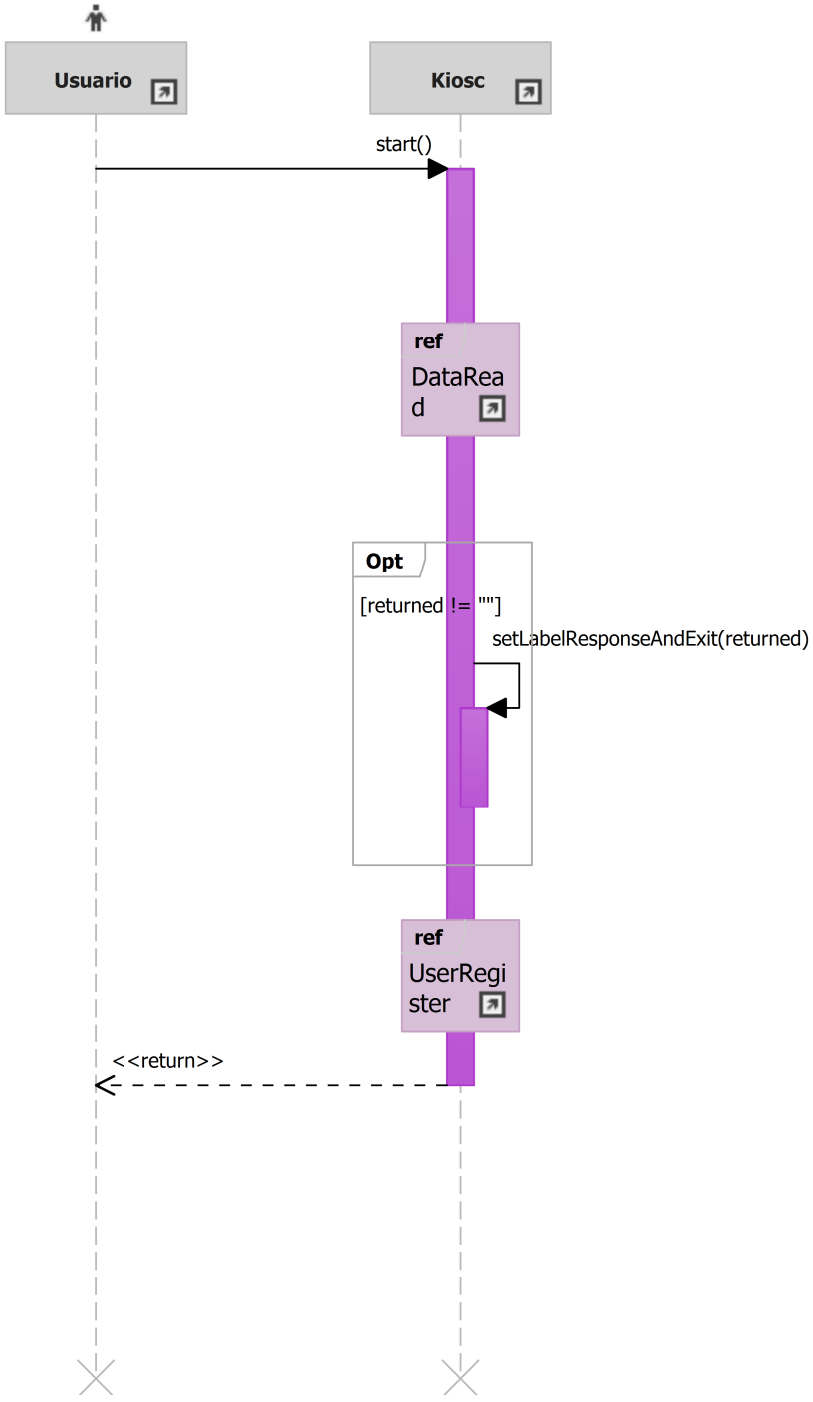


Ilustración 74: sd Llamar.

- 02- Registro del usuario. Identificador del diagrama: **sd RegistroUsuario.**

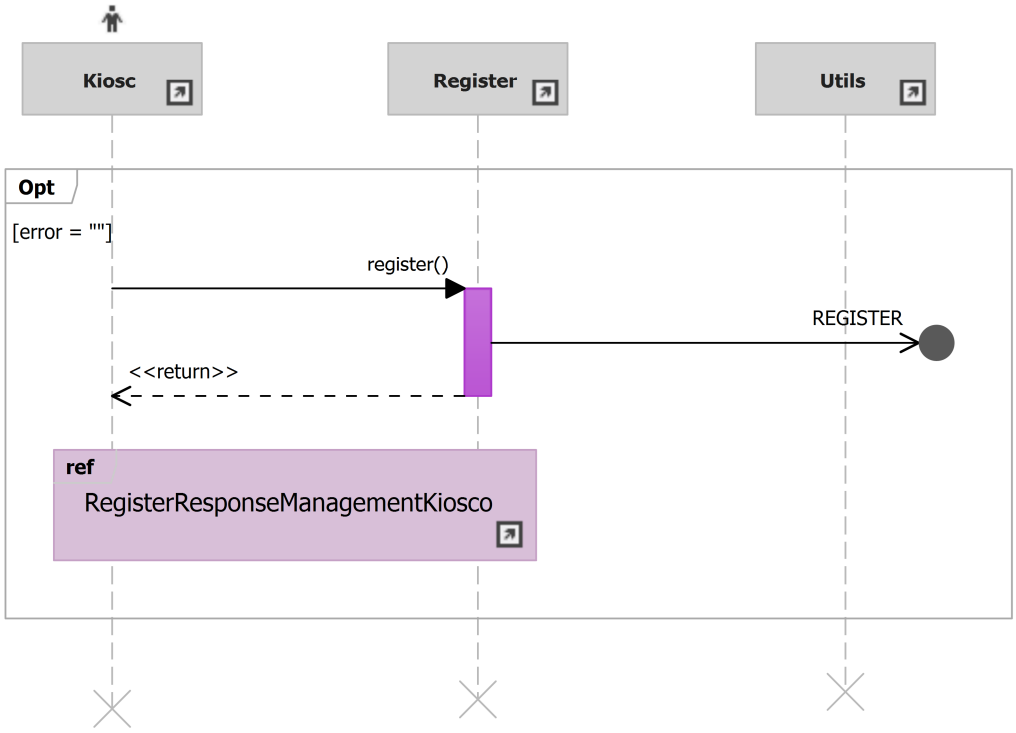


Ilustración 75: sd UserRegister.

- 03- Lectura de datos. Identificador del diagrama: **sd DataRead**.

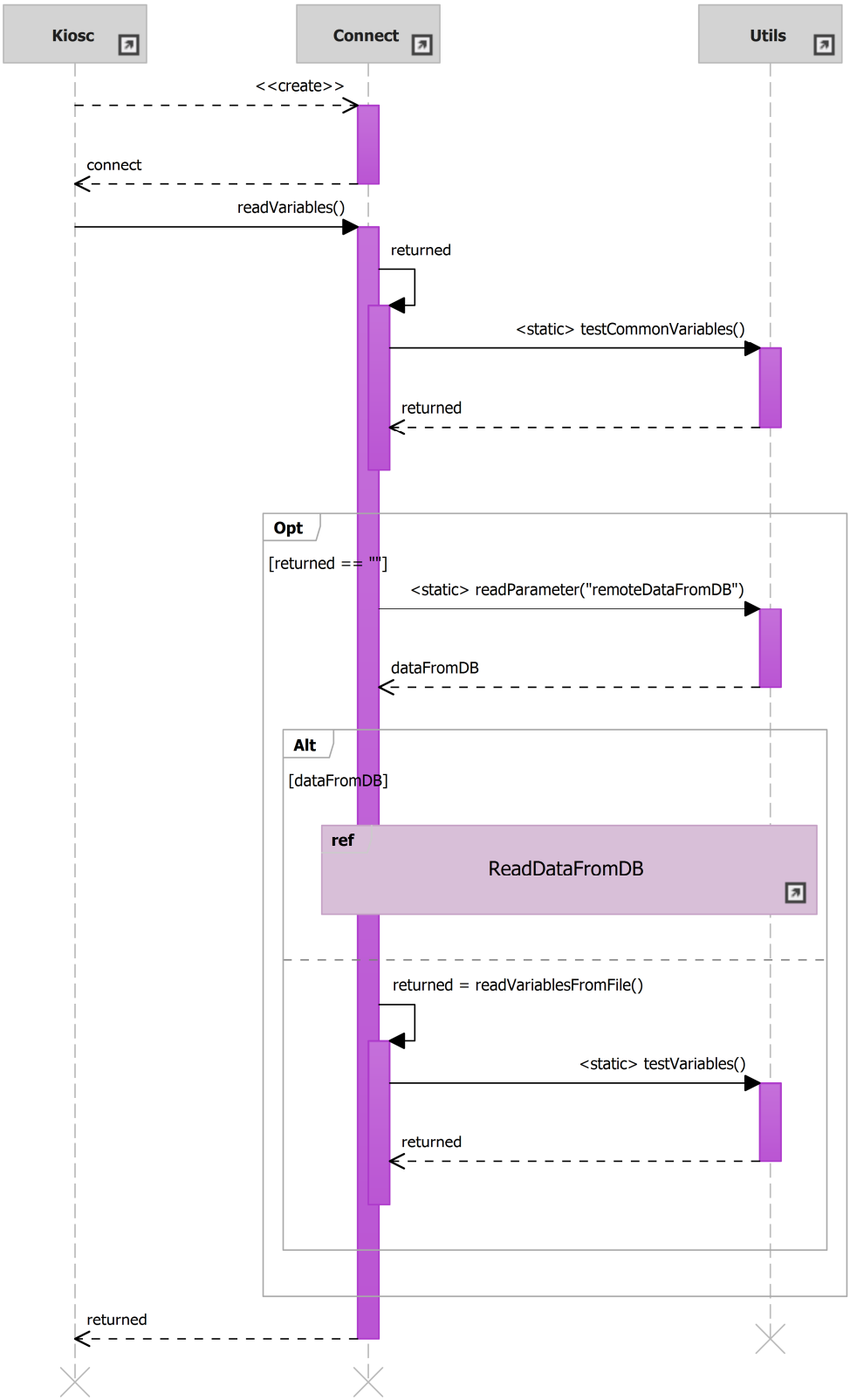


Ilustración 76: sd DataRead.

- 04- Lectura de datos desde el servidor. Identificador del diagrama: **sd ReadDataFromDB**.

Ilustración 77: sd ReadDataFromDB.

- 05- Cancelar llamada. Identificador del diagrama: **sd CancelarLlamada**.

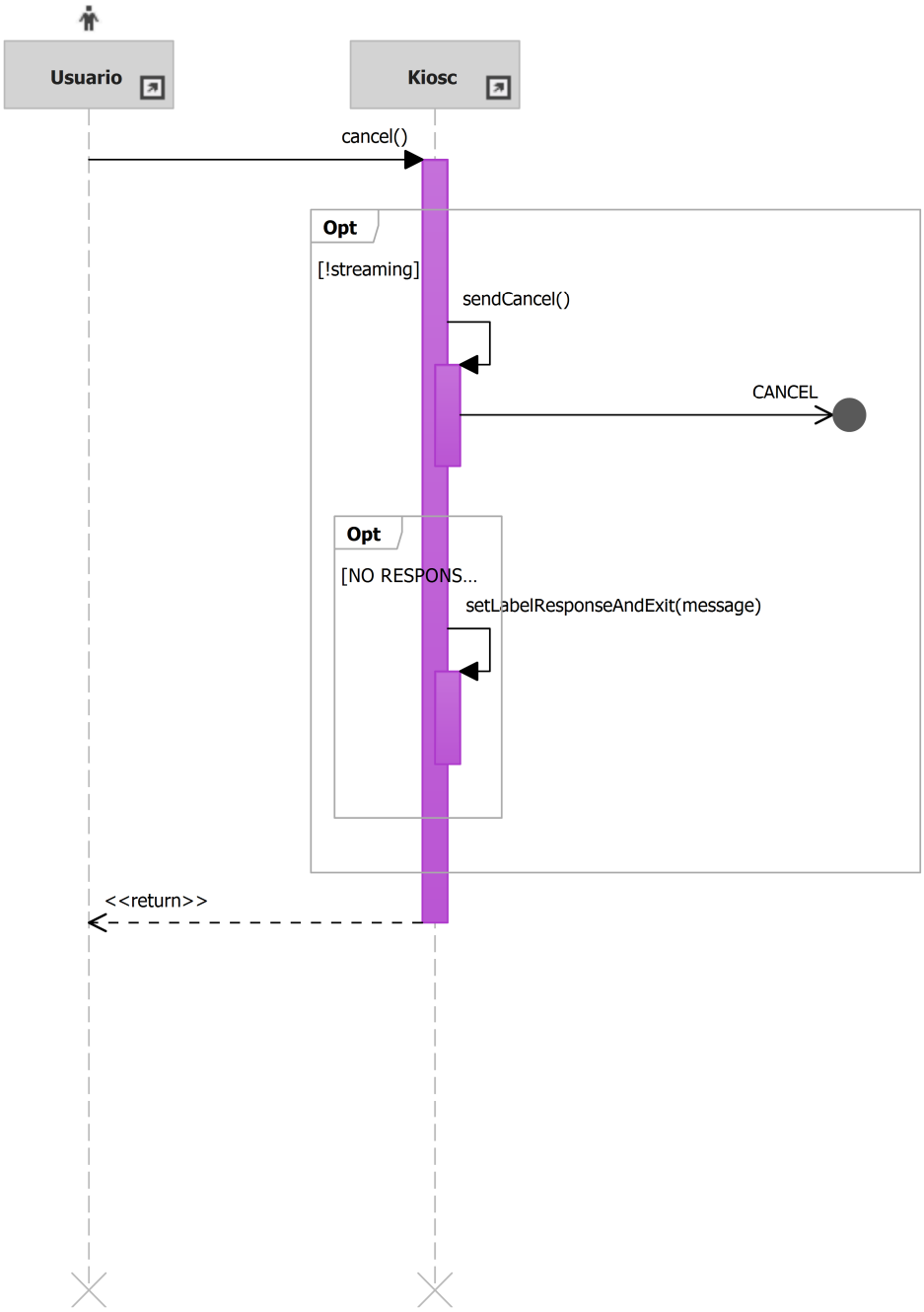


Ilustración 78: sd CancelarLlamada.

- 18- Empezar llamada. Identificador del diagrama: **sd EmpezarLlamada**.

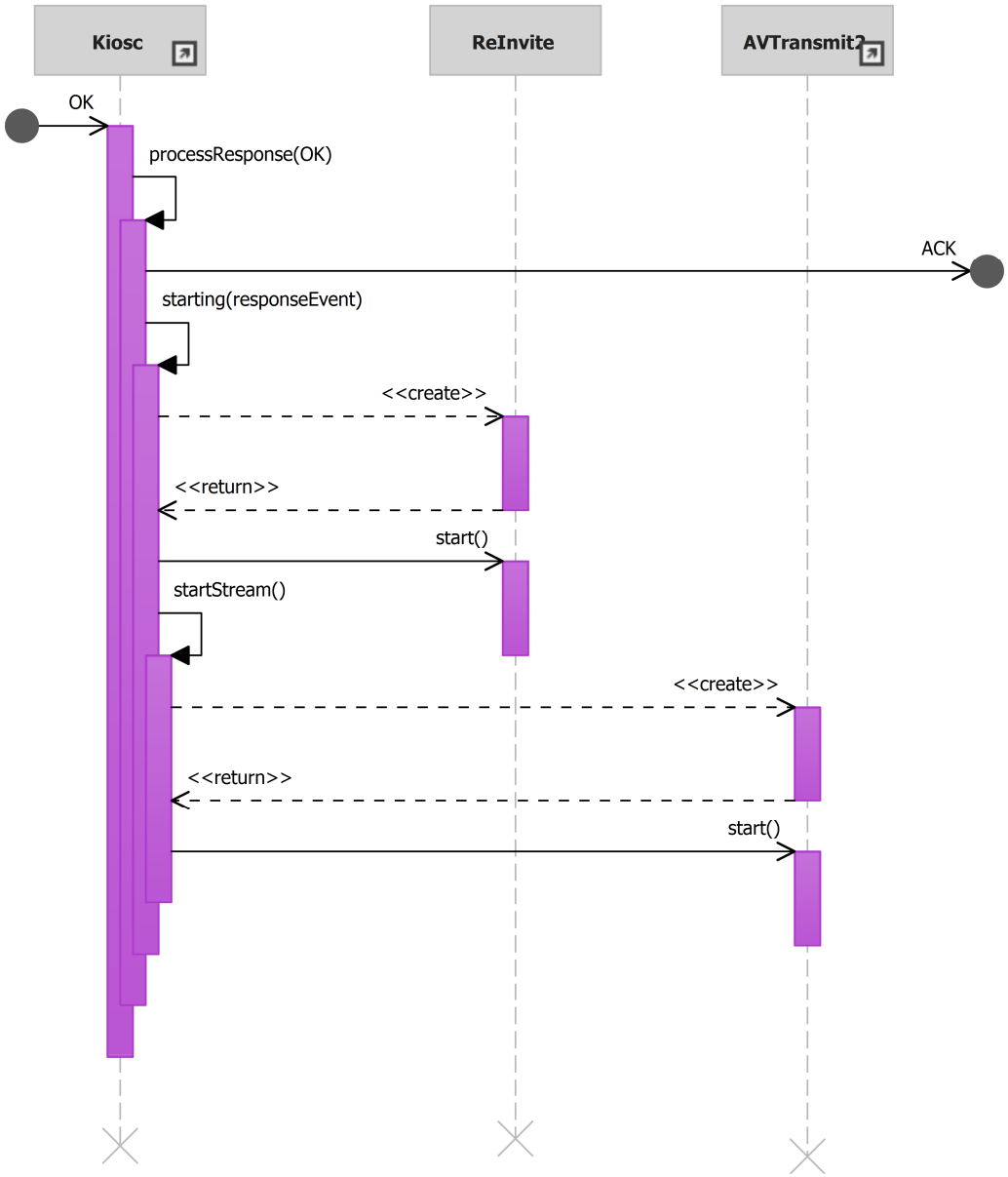


Ilustración 79: sd EmpezarLlamada.

- 19-Tratar rechazo llamada. Identificador del diagrama: **sd TratarRechazoLlamada**.

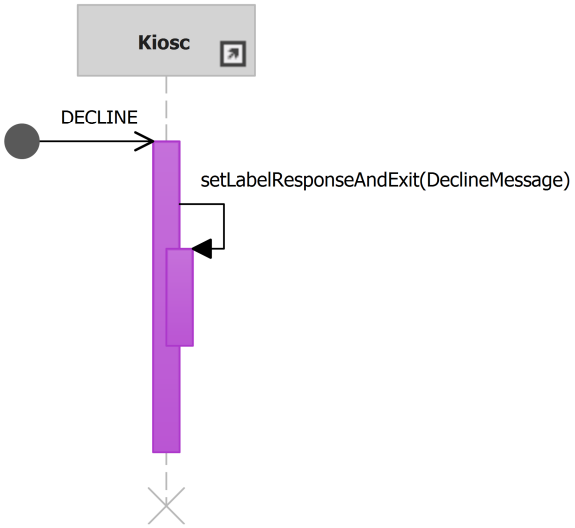


Ilustración 80: sd TratarRechazoLlamada.

- 20- Esperar. Identificador del diagrama: **sd Esperar**.

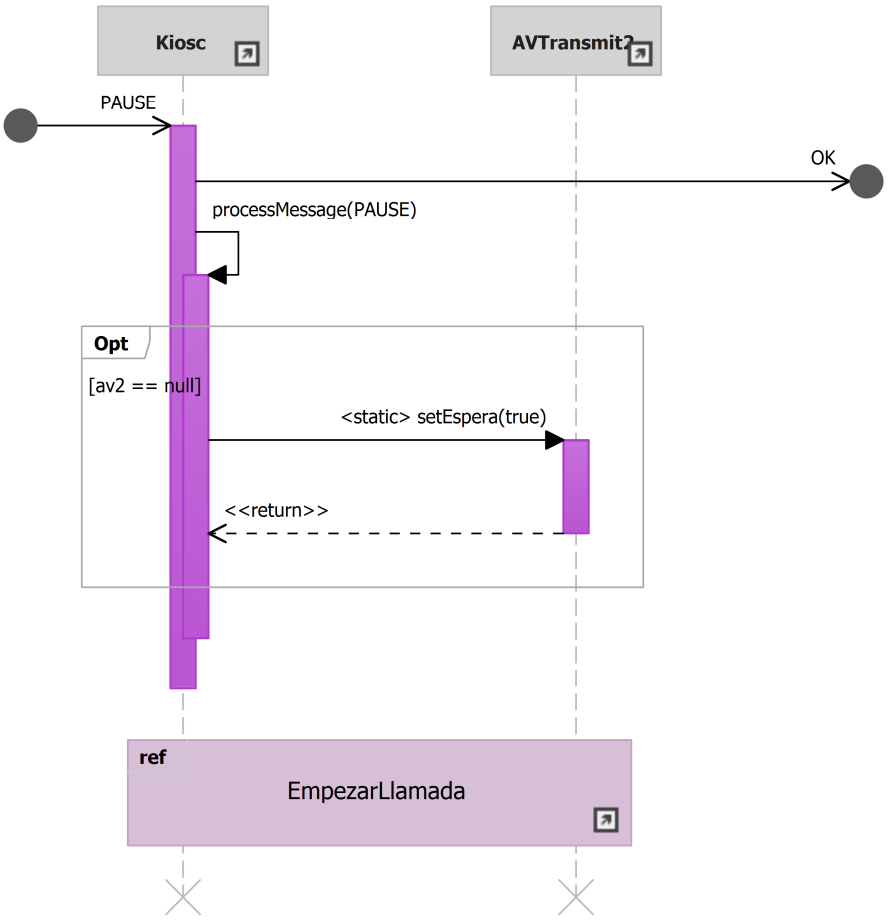


Ilustración 81: sd Esperar.

- 21- Mostrar escritorio. Identificador del diagrama: **sd MostrarEscritorio.**

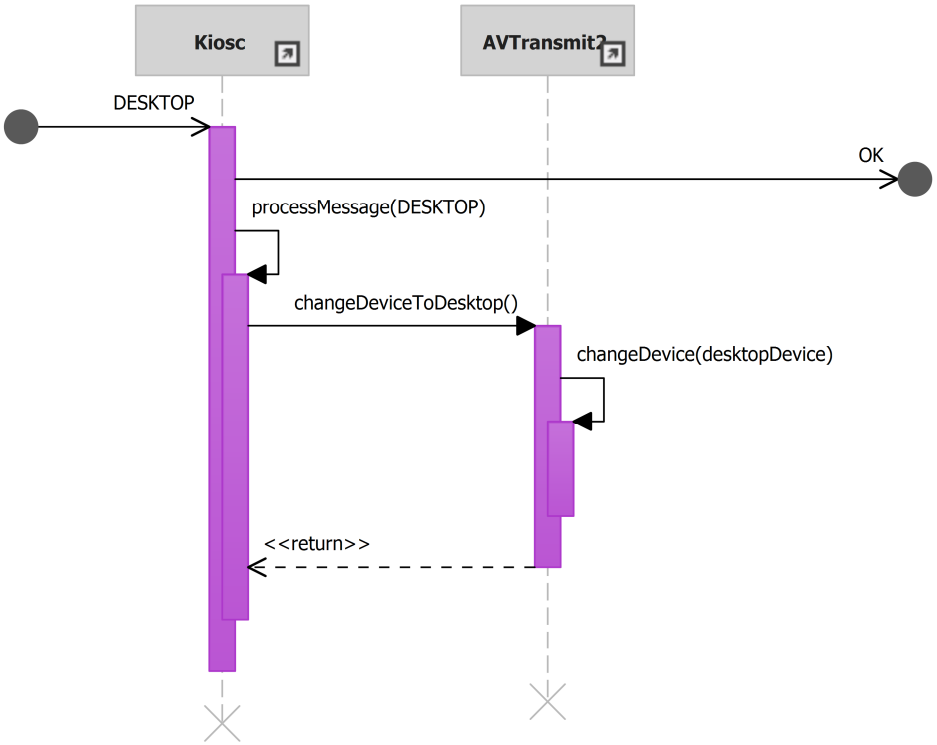


Ilustración 82: sd MostrarEscritorio.

- 22- Activar visión cámara. Identificador del diagrama: **sd ActivarVisionCamara**.

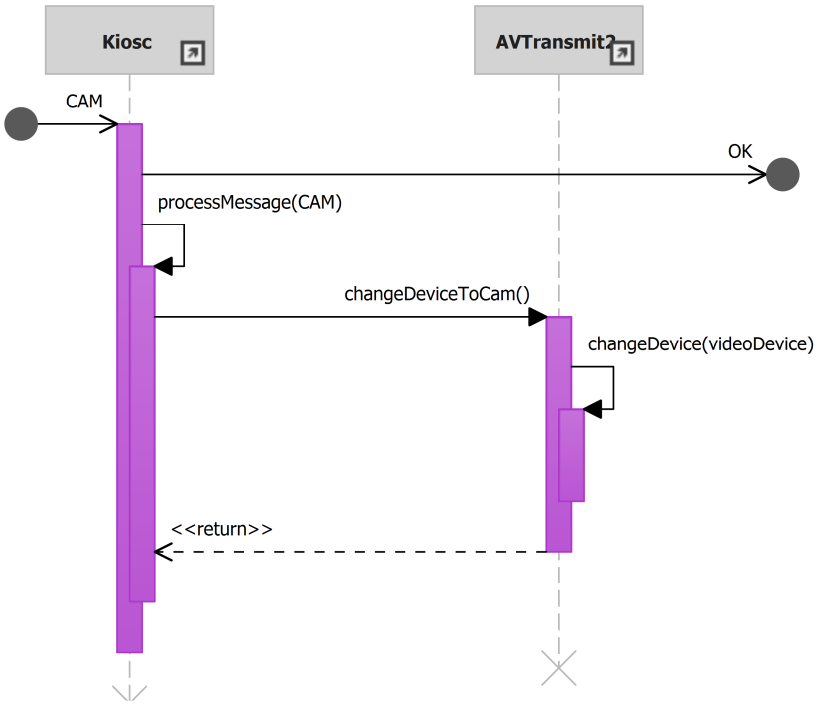


Ilustración 83: sd ActivarVisionCamara.

- 23- Activar llamada. Identificador del diagrama: **sd ActivarLlamada**.

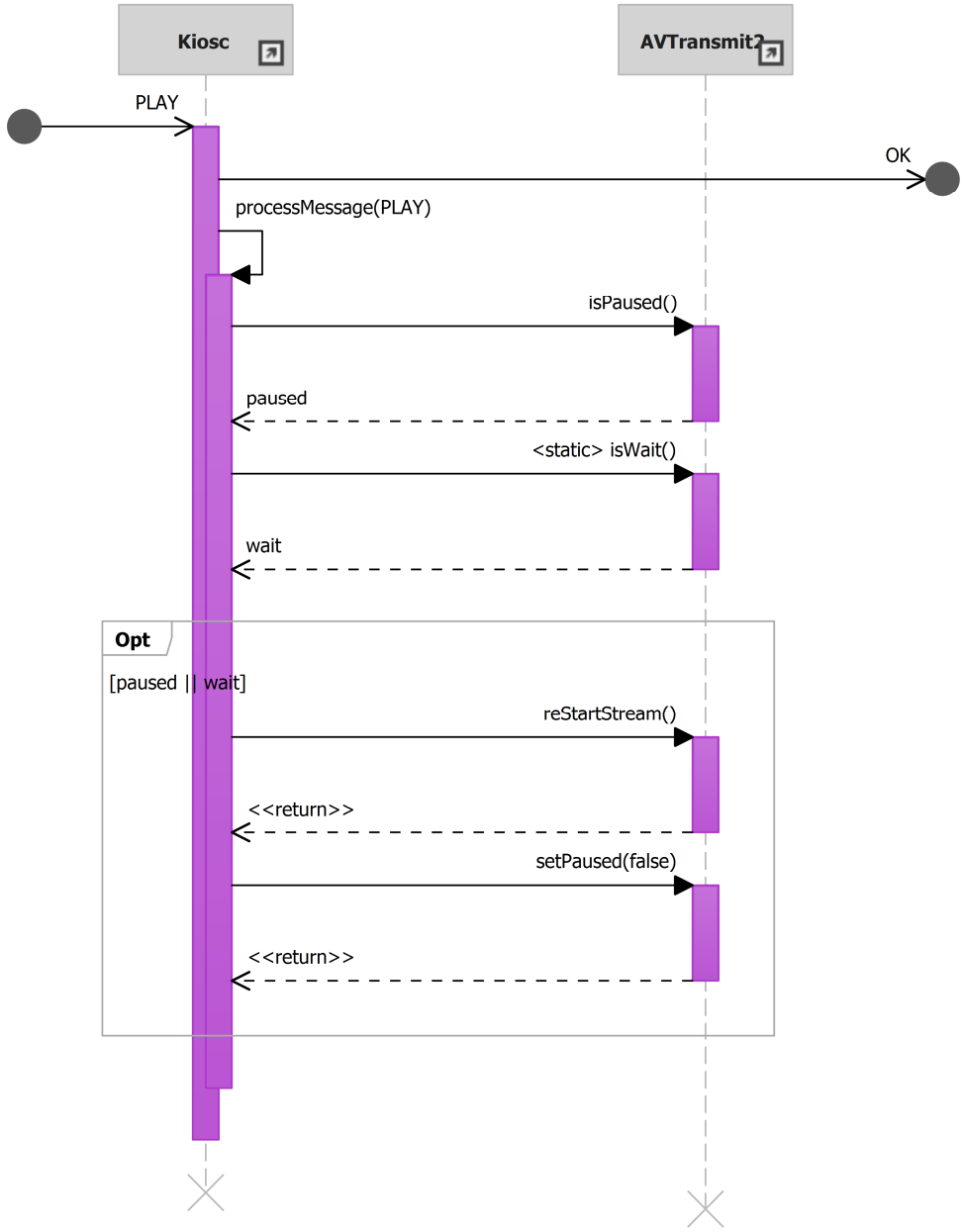
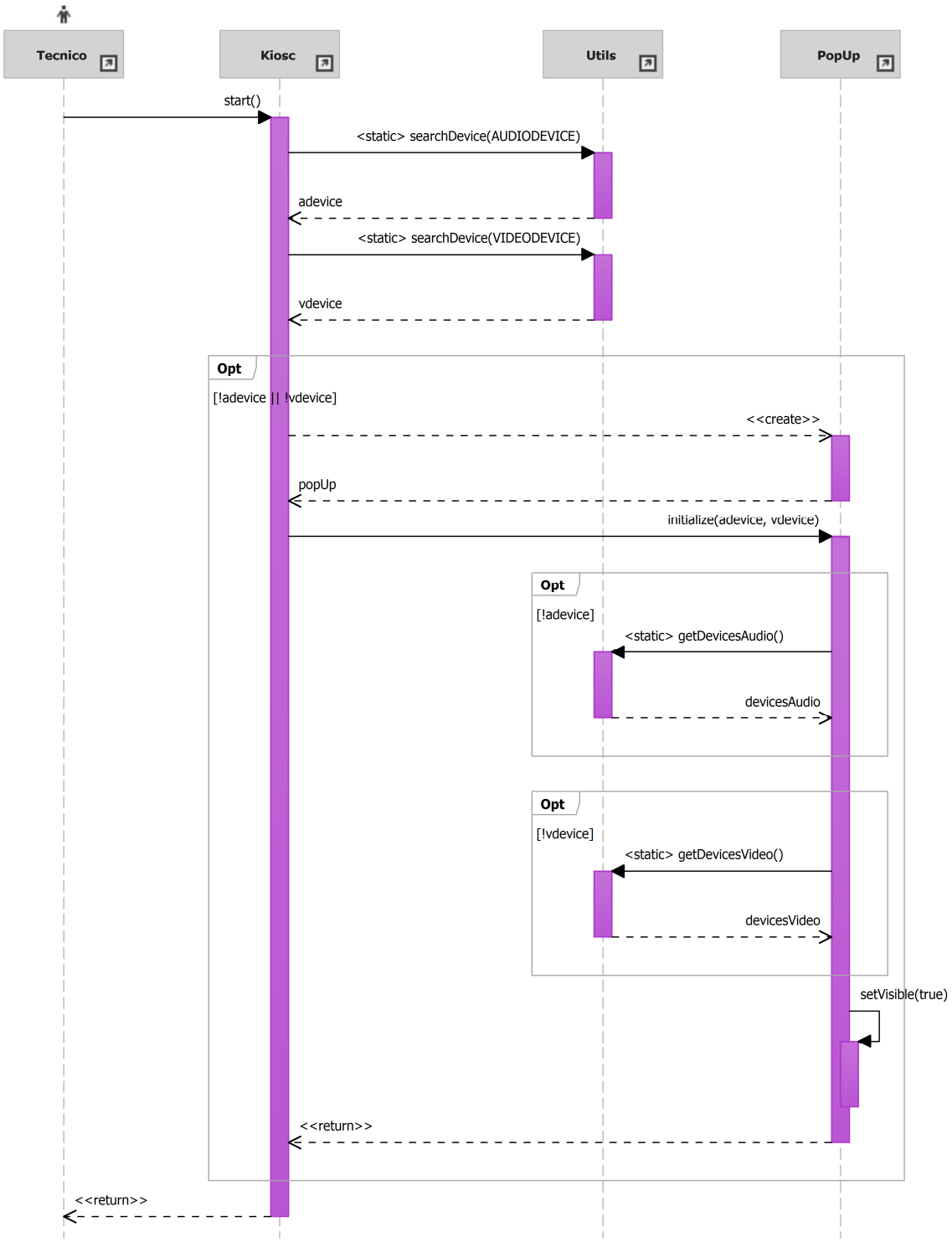


Ilustración 84: sd ActivarLlamada.

- 24,25- Predeterminación dispositivo de audio, video. Identificador del diagrama: **sd PredDisp**.



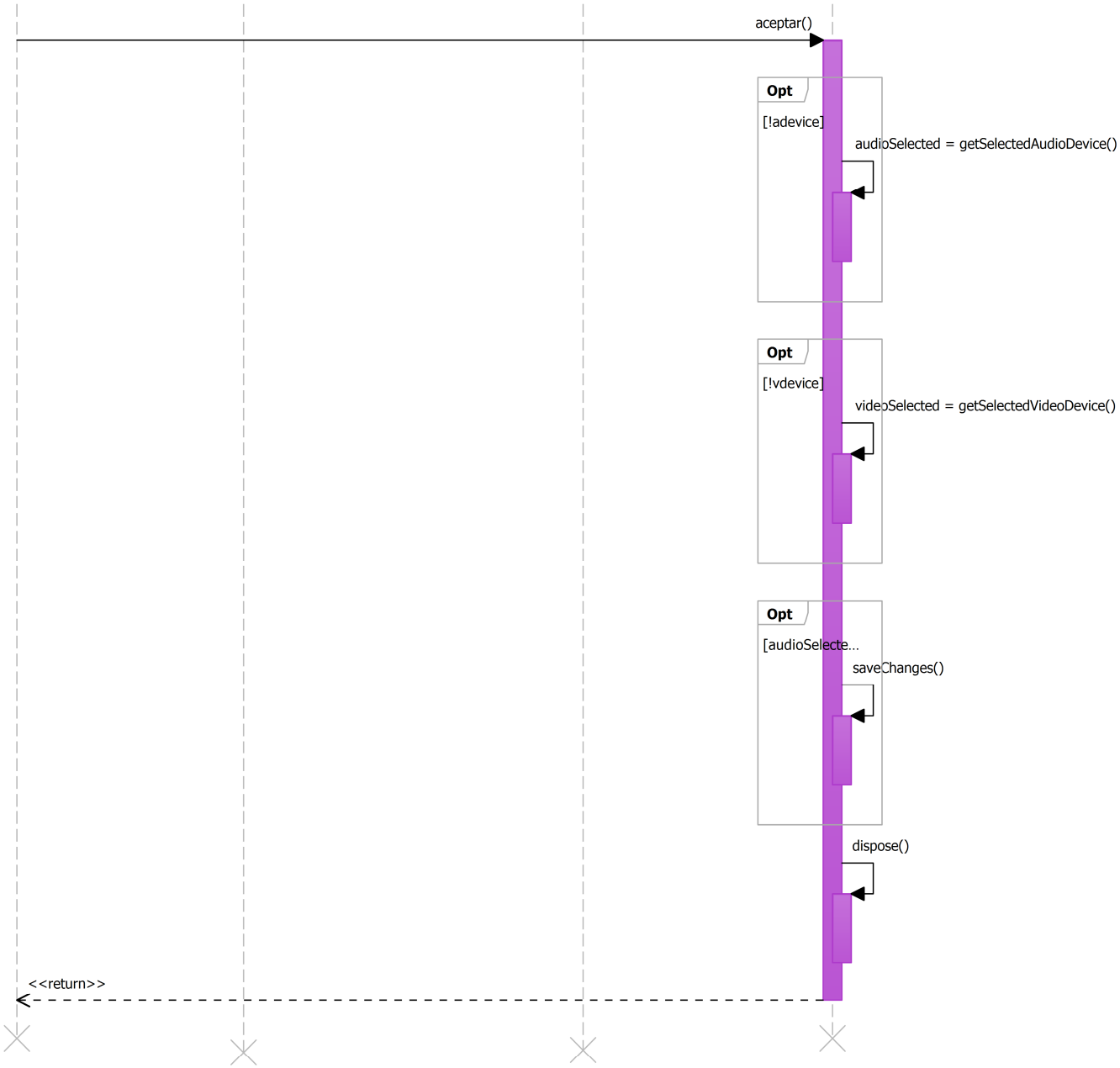


Ilustración 85: sd PredDisp.

- 26- Pausar llamada asistente. Identificador del diagrama: **sd PausarLlamadaAsistente**.

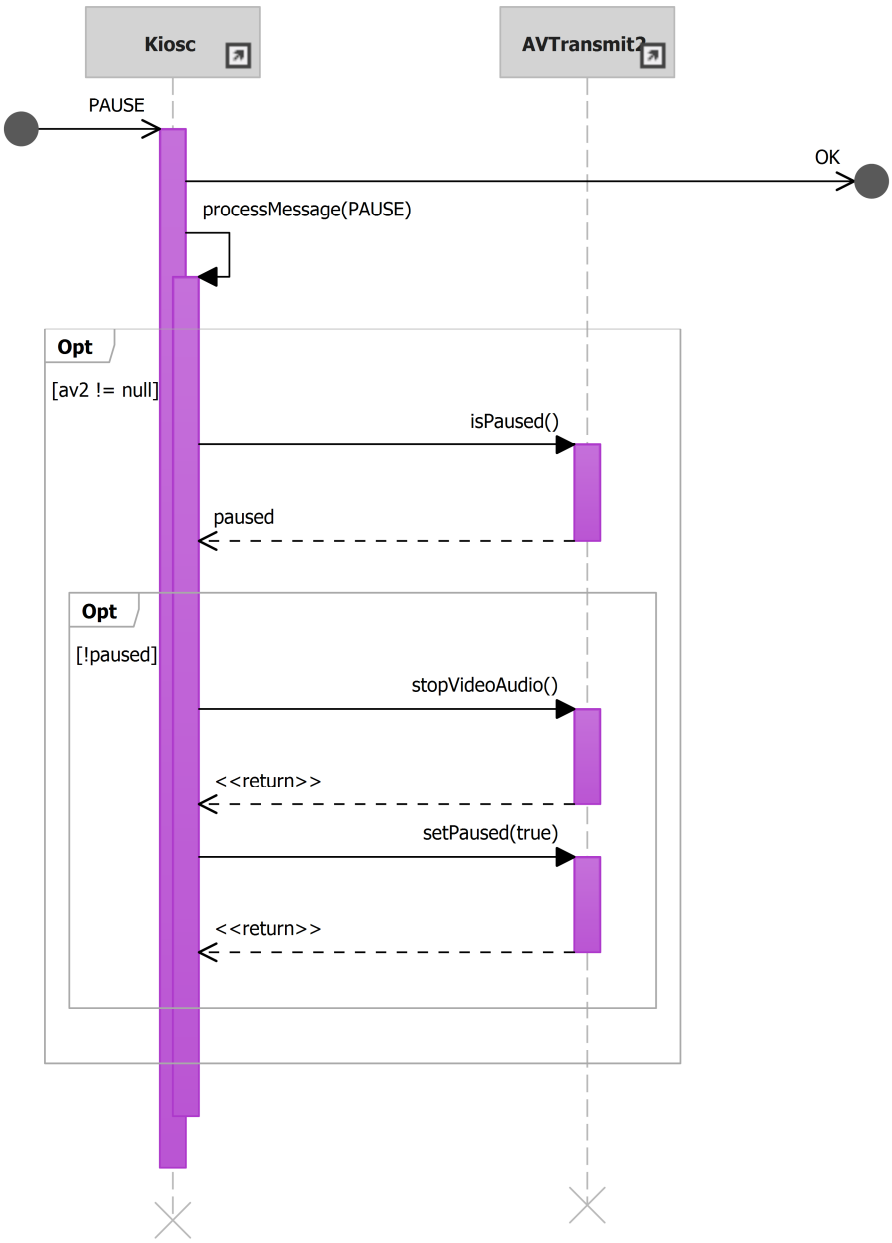


Ilustración 86: sd PausarLlamadaAsistente.

- 27- Finalizar llamada asistente. Identificador del diagrama: **sd FinalizarLlamadaAsistente**.

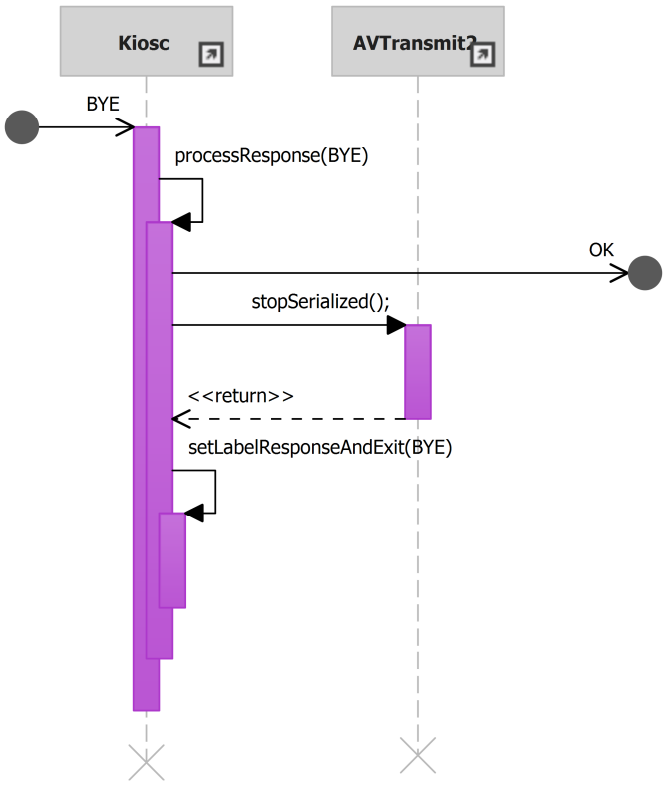
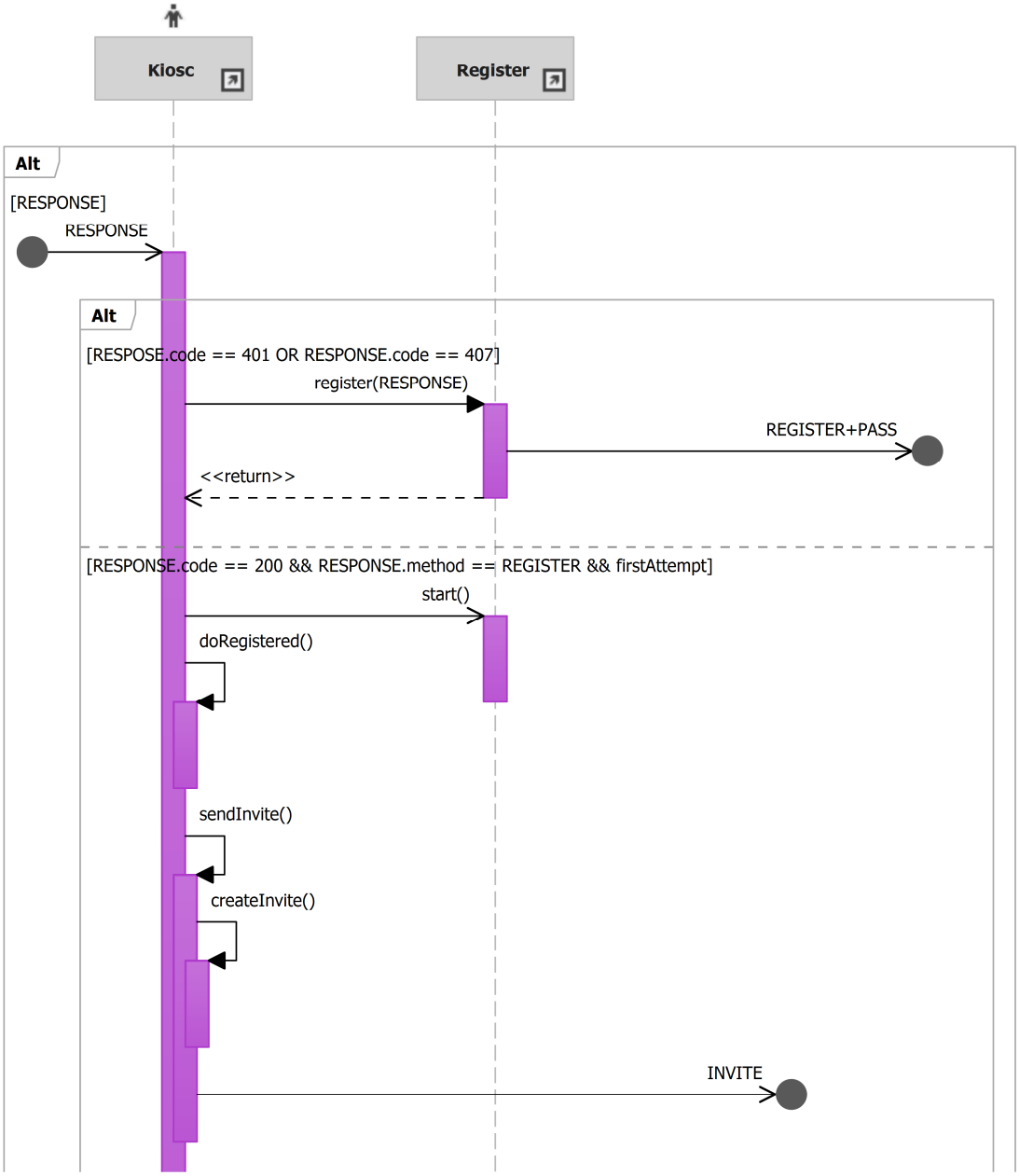


Ilustración 87: sd FinalizarLlamadaAsistente.

Diagramas auxiliares

- sd RegisterResponseManagementKiosco.



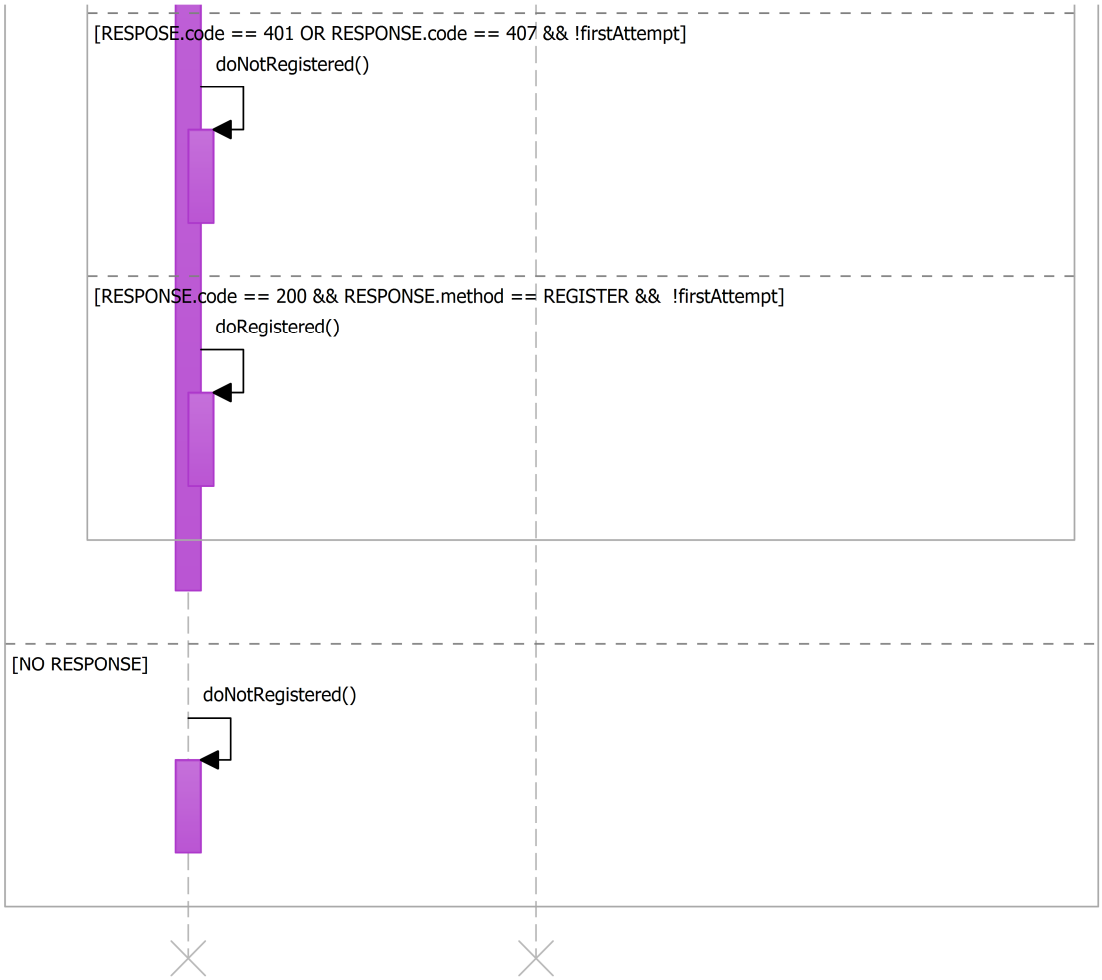


Ilustración 88: sd RegisterResponseManagementKiosco.