Sistema de Notificaciones para la Plataforma de Desarrollo e Integración Continua de Nova.

Autor

Jesús Rabelo Pérez.

Tutores

Dariem Pérez Herrera

Adrian Crúz Machín

Ciudad de la Habana

Mayo 2014

Declaración de Autoría

Declaro que soy el único autor del presente trabajo y autorizo a la Facultad 1 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los ____ días del mes de ____ del año _____.

_____	_____
Dariem Pérez Herrera	**Adrián Crúz Machín**

Agradecimientos

- *A mis familiares por el apoyo incondicional durante estos 5 años de carrera.*
- *A mi novia por su compañía en los momentos más difíciles.*
- *A Fidel por su visión de crear la Universidad de las Ciencias Informáticas.*
- *A mis profesores y amigos de clases por mi formación y por soportarme durante estos 5 años.*
- *A mis tutores y compañeros de proyecto por todo el apoyo brindado.*

Dedicatoria

A mi mamá Janet Pérez Rodrigues y a mi papá Fernando Rodrigues Rodrigues.

Ustedes son las luces que me guían en este mundo de oscuridad.

Resumen

Actualmente las herramientas de desarrollo Wana-Build, Soyuz, Koji, Open Build Service y su integración en diferentes plataformas web como la ya muy conocida Launchpad han jugado un papel decisivo en el éxito de distribuciones punteras como Ubuntu, Debian, Fedora y openSUSE respectivamente. Gracias a sus funcionalidades, los desarrolladores de las empresas o comunidades, según sea el caso, solo tienen que preocuparse por tener el código fuente listo para enviarlo, y la infraestructura se encarga de construir la distribución casi de forma automática, alcanzando los diferentes sistemas de notificación que la componen, un reconocido protagonismo.

La presente investigación tiene como objetivo implementar un sistema de notificaciones, en la Plataforma de Desarrollo e Integración Continua de la distribución cubana de GNU/Linux, para notificar a sus usuarios de manera eficaz y escalable de los eventos asociados a las herramientas que intervienen en el proceso de construcción. Realiza un estudio sobre los protocolos y *middleware* para el paso de mensajes, con mayores perspectivas en la actualidad. Expone los aspectos de la solución propuesta, y realiza la verificación de su escalabilidad y eficacia. Su principal resultado lo constituye la obtención de un sistema funcional, compuesto por un generador de notificaciones y una aplicación cliente que se incrusta en la barra de notificación de las distribuciones Nova, manteniendo informado a los usuarios de la plataforma mediante un mecanismo de publicación/suscripción. Además de una arquitectura que soporta la incorporación de diferentes vías de comunicación fácilmente, garantizando escalabilidad y eficacia.

Palabras claves: plataformas de desarrollo e integración continua, sistema de notificaciones, nova, Icono de Notificación, mensajes, *middleware*.

Índice

Introducción

El desarrollo acelerado de las distribuciones de GNU/Linux se ha mantenido como uno de los temas más activos y en constante evolución. Hoy a más de 20 años del surgimiento de la primera de ellas, prácticamente no hay lugar donde no se encuentren presente. Ordenadores personales, teléfonos móviles, tabletas y electrodomésticos, son algunas de las formas más comunes en las que se manifiestan [Llamaret, 2013].

Actualmente Debian, Ubuntu, Fedora y openSUSE se encuentran entre las distribuciones más destacadas [Llamaret, 2013]. La calidad de su paquetería y la agilidad para obtener versiones nuevas o mejoradas de cada uno de sus productos, ha estado muy ligado a las infraestructuras tecnológicas de sus empresas. Gracias a los servidores que las componen y el software que corre en ellos, los desarrolladores solo tienen que preocuparse de tener el código fuente listo y enviarlo [Herrera, 2013]. Wana-Build, Soyuz, Koji y Open Build Service, son ejemplos respectivos de herramientas que conforman dichas infraestructuras [Hodek, 2013]. Sus diferentes componentes agilizan la construcción de paquetes y la gestión de repositorios. Automatizan todo el proceso de elaboración de la imagen y permiten detectar fallos de seguridad. Facilitan la liberación de aplicaciones, actualizaciones, extensiones, e incluso distribuciones completas. Todas en un amplio rango de arquitecturas de sistemas operativos y *hardware* al servicio de la comunidad [Herrera, 2013].

Un aspecto importante para dichas infraestructuras y la concepción de sus productos, ha sido la evolución de la informática y las comunicaciones. Sin el surgimiento del Internet, el correo electrónico, la telefonía móvil y los clientes de mensajería instantáneas, el éxito no sería el mismo. Es por eso que muchos componentes incluyen interfaces web o se apoyan en sus wiki[1], las cuales poseen sistemas de notificaciones por distintas vías, garantizando que los usuarios finales participen y estén al tanto de todo lo que acontece [Edwards, 2011]. Dichos sistemas se apoyan a su vez en diferentes productos que han buscado dominar el campo de la mensajería eficiente, contando con gran relevancia los *middleware* orientados a mensajes (MOM).

Los MOM son aplicaciones que utilizan los mensajes como método de integración y provee mecanismos para crearlos, almacenarlos y trasmitirlos. Se ubican entre la capa más alta (usuarios y aplicaciones) y la más baja (mecanismos de comunicación básicos) [Pressman, 2009]. Utilizan diversos estándares de protocolos abiertos para la comunicación, entre los que destaca el protocolo orientado a texto simple o en flujo (del inglés STOMP), el servicio de mensajería de java (del inglés, JMS) y el protocolo avanzado de

[1] Sitio web donde se presenta toda la información relacionada con un producto específico.

espera de Mensajes (del inglés, AMPQ), el cuál se considera el estándar de facto para este tipo de *middleware*. Hoy son ampliamente utilizados en entornos militares, tele-vigilancia, aeronáutico, aeroespaciales y en empresas prestigiosas como Google, Mozilla y AT&T [Videla, 2012].

Cuba desde febrero del 2009 cuenta con Nova, una distribución cubana de GNU/Linux que busca minimizar la dependencia tecnológica [Pierra, 2013], desarrollada y mantenida por el Departamento de Sistema Operativo (DSO) del Centro de Soluciones Libres (CESOL) de la Universidad de las Ciencias Informáticas (UCI) [Oramas, 2013]. Desde su surgimiento ha pasado por diferentes etapas que han evidenciado la ineficacia de su infraestructura [Herrera, 2014]. Los problemas que han presentado las herramientas y métodos que componen su desarrollo base, han contribuido a que Nova sea reconocida ante el mundo como una distribución descontinuada [DistroWatch.com, 2014]. Considerando como las distribuciones exitosas integran sus herramientas, y como la interacción entre usuarios y desarrolladores repercute en su calidad. Los integrantes del DSO decidieron concebir su propia plataforma, en busca de una infraestructura más estable y eficiente.

La Plataforma de Desarrollo e Integración Continua de Nova, cuenta con un Sistema de Gestión de Repositorios[2], un Sistema Distribuido de Compilación[3], un Sistema para la Generación Automática de Imágenes[4] de CD/DVD y una herramienta de escaneo automático de código fuente[5] [Herrera, 2013]. Sin embargo en un ambiente donde las operaciones pueden tardar horas en completarse, los usuario se ven obligados ha permanecer en línea con ella o estar pendiente con cierta frecuencia, para conocer si sus acciones han sido completadas o se han interrumpido por la ocurrencia de errores, lo cual genera molestos contratiempos que desmotivan y afectan el desarrollo de Nova. Dado que no se gestionan los eventos emitidos por dichas operaciones, se dificulta además el conocimiento de las actualizaciones, fallos detectados y mejoras realizadas a la distribución cubana o su paquetería por los demás miembros de las comunidad, provocando duplicación de esfuerzo y pérdida de tiempo en ajustes o desarrollos ya efectuados por otros. Teniendo en cuenta lo que significa para el éxito de las distribuciones actuales tener una comunidad que la respalde y contribuya constantemente con ideas nuevas, se hace urgente la búsqueda de una solución que permita mantenerla informada con niveles aceptables de escalabilidad y eficacia, que a intereses de la presente investigación están dados por los siguientes aspectos.

Eficacia:

[2] Sistema que gestiona los repositorios de paquetes tanto de código fuente como binario de las distintas variantes de la distribución cubana de GNU/Linux.
[3] Se encarga de la generación de paquetes binarios a partir de código fuente mediante una granja de computadoras.
[4] Automatiza el proceso de generación de imágenes en formato ISO 9660 para ser quemadas en CD/DVD.
[5] Herramienta elaborada por los integrantes del proyecto Nova para la detección de fallas de seguridad en código fuente.

- **Resistencia a fallos:** El tratamiento de los mensajes al producirse un fallo técnico tanto en el servidor como en el cliente trae consigo menos mensajes/notificaciones perdidos, y por tanto los usuarios presentan una menor desinformación [Offut, 2002].

- **Tiempo de respuesta:** Cuanto menor sea el tiempo de respuesta al crear y enviar mensajes/notificaciones una vez que se produce un evento o acción, los usuarios se notifican más próximo a la culminación real del evento o acción [Offut, 2002].

2. Escalabilidad:

- **Adaptabilidad a un número de peticiones cada vez mayor:** La forma en que se elimine las brechas de comunicación entre diferentes arquitecturas, plataformas y sistemas operativos, permite atender mayor o menor número de peticiones sin que se pierda o cese por completo la capacidad de respuesta [Pressman, 2009].

Por tanto, en términos de pregunta de investigación **el problema** se formula de la siguiente forma: ¿Cómo notificar a los usuarios de la Plataforma de Desarrollo e Integración Continua de Nova, sobre los eventos de su interés, de manera eficaz y escalable?

Estableciendo como **objeto de estudio,** el proceso de notificación y mensajería mediante *middleware* orientados a mensajes, se plantea entonces, como **objetivo general** de la presente investigación: Desarrollar un sistema de notificaciones, de manera eficaz y escalable, para la suscripción de los usuarios a temas de su interés, en la Plataforma de Desarrollo e Integración Continua de Nova, tomando como **campo de acción**, el proceso de notificación y mensajería en la Plataforma de Desarrollo e Integración Continua de Nova.

Con el propósito de alcanzar el objetivo general se plantean los siguientes **objetivos específicos**:

1) Analizar las distintas herramientas para el paso de mensajes y protocolos de notificación que existen y seleccionar las más adecuada para la solución.

2) Diseñar la arquitectura del sistema de notificaciones para la Plataforma de Desarrollo e Integración Continua de Nova.

3) Implementar el sistema de Notificaciones para la Plataforma de Desarrollo e Integración Continua de Nova.

4) Realizar pruebas de estrés y carga para comprobar la eficacia y escalabilidad del sistema de notificaciones de la Plataforma de Desarrollo e Integración Continua de Nova.

Una vez profundizado en las diferentes herramientas y protocolos informáticos para el paso de mensajes se formula la siguiente **idea a defender:** El desarrollo del sistema de notificaciones de la Plataforma de Desarrollo e Integración Continua de Nova, permitirá la suscripción de los usuarios a temas de su interés, de manera eficaz y escalable.

Para el desarrollo de la solución propuesta se plantean las siguientes tareas de investigación:

1) Estudio del marco de trabajo web *Django* y el código fuente de la Plataforma de Desarrollo e Integración Continua de Nova, para la implementación de su sistema de notificaciones.

2) Estudio de las distintas herramientas y protocolos existentes para el paso de mensajes.

3) Desarrollo de un módulo de *Django* y de un cliente de escritorio para la conformación del sistema de notificaciones de la Plataforma de Desarrollo e Integración Continua de Nova.

4) Elaboración de casos de pruebas para verificar la escalabilidad y eficacia del sistema de notificaciones de la Plataforma de Desarrollo e Integración Continua de Nova.

Para dar cumplimiento al objetivo propuesto, se han combinado diferentes métodos teóricos y empíricos de la investigación científica, en la búsqueda y procesamiento de la información:

Métodos teóricos: El método histórico-lógico para el estudio de los trabajos anteriores que constituyen referentes teórico-prácticos en el tratamiento del problema planteado, así como el analítico-sintético para recopilar la información de las diferentes bibliografías relacionadas con los sistemas de notificaciones, y avanzar hacia la solución.

Métodos empíricos: El método experimental para elaborar los casos de pruebas y proceder a la comprobación de los indicadores que definen la escalabilidad y eficacia de la solución propuesta, para el problema planteado.

Capítulo 1: Análisis de los protocolos y middleware orientados a mensajes

Con el objetivo de facilitar la comprensión y establecer un punto de partida hacia la solución propuesta, en el presente capítulo se exponen conceptos fundamentales asociados al dominio del problema planteado. Se realiza un análisis profundo sobre los principales protocolos y *middleware* orientados a mensajes y se fundamenta la selección de los más adecuados para el desarrollo de la solución. También se especifican detalles sobre las posibles herramientas y tecnologías para el cumplimiento del objetivo.

1.1 Conceptos asociados al dominio del problema

Según la Real Academia de La Lengua Española, "*las **notificaciones** son acciones o efectos de comunicar* [Rae, 2013]. Siendo los **sistemas de notificaciones** aplicaciones que generan y envían mensajes a sus usuarios por un canal específico, manteniéndolos informados en todo momento".

La **comunicación asíncrona** consiste en la posibilidad de postergar la entrega de mensajes hasta que se estime conveniente, mientras que la **comunicación síncrona** se refiere a la entrega de mensajes de manera inmediata [Pressman, 2009].

La **escalabilidad** en informática y comunicaciones es la capacidad que presenta un sistema para escalarse y manejar 100, 1000, 10000 usuarios, sin que se pierda o cese por completo la capacidad de respuesta [Pressman, 2009].

La **eficacia** en informática y comunicaciones es la capacidad que tiene un sistema de cumplir con los objetivos para los que fue creado [Offut, 2002].

Un **protocolo de comunicación** en informática y comunicaciones, se refiere a un conjunto de reglas y estándares, que controlan la secuencia de mensajes entre entidades que conforman una red, por ejemplo teléfonos y computadoras.

El **correo electrónico** es un servicio de red que permite a los usuarios enviar y recibir mensajes. Entre los protocolos más comunes para el envío y recepción de mensajes se encuentran: El Protocolo de Oficinas en su tercera versión (del inglés POP3), el protocolo de acceso a Internet (del inglés IMAP) y el protocolo simple de trasmisión de correo (del inglés SMTP) [Sivianes, 2010].

La **Sindicación Realmente Simple** (del inglés RSS) hace referencia al sistema completo mediante el cual una página web emite sus principales noticias e informaciones. Las noticias son emitidas a través de los canales RSS y son recibidas por los lectores RSS [Sivianes, 2010].

Los **lectores RSS** es el programa que permite a los usuarios suscribirse a todos los artículos o noticias de su interés en sus páginas web o blog favoritos. Reúnen en un solo lugar todos los titulares de las páginas a las que se ha suscrito el usuario [Sivianes, 2010].

Los **canales RSS** son la forma en que las noticias llegan al lector RSS. Las noticias de las páginas web llegarán por el canal RSS al ordenador y el lector RSS se encargará de mostrarlas [Sivianes, 2010].

La **telefonía móvil** o telefonía celular como también es conocida es un servicio que para la sociedad de nuestros días se ha vuelto indispensable. Millones de personas del mundo entero se comunican diariamente ya sea a través del servicio de voz o por mensajes de texto.

1.2 Sistemas de notificaciones

Para brindar solución al problema planteado sobre la necesidad de mantener informados a los usuarios de la Plataforma de Desarrollo e Integración Continua de Nova con una alta escalabilidad y eficacia. La presente investigación presupone el estudio de algunos de los sistemas de notificaciones, que permiten la comunicación por diferentes servicios con numerosos usuarios. La red social Facebook, contiene un sistema de notificación basado fundamentalmente en el envío de mensajes por correo electrónico, Jabber[6] y telefonía móvil, que posibilita a sus usuarios, conocer toda la información que se mueve alrededor de sus cuentas.

Twitter utiliza la biblioteca de *sockets*[7] ZeroMQ, para el envío eficiente de mensajes cortos a correo electrónico y teléfonos móviles. La plataforma colaborativa de software libre Launchpad, presenta un sistema de notificación configurable, con listas de correo electrónicos, a las cuales los usuarios se suscriben de acuerdo a su preferencia [Launchpad, 2014]. En el proyecto Fedora se utilizan además los canales IRC (Internet Relay Chat) como comunicación inmediata [fedoraproject.org, 2014]. Cubadebate contiene un sistema de notificación basado en un canal RSS, que envía a los usuarios suscritos por correo electrónico un resumen de las noticias más relevantes del día [Alonso, 2009].

1.3 Los Middleware

Los *middleware* son una capa intermedia, ubicada entre la capa más alta representada por los usuarios y aplicaciones, y la capa más baja que son los sistemas operativos y mecanismos de comunicación básicos. Provee una interfaz común y por tanto un mismo medio de comunicación para todas las aplicaciones. Oculta las diferencias de hardware, lenguaje de programación, plataformas de comunicación y sistemas

[6] Servicio de mensajería instantánea o chat como normalmente se le conoce.
[7] Constituyen un concepto abstracto por el cual dos programas pueden intercambiar cualquier flujo de datos.

operativos, lo que facilita la implementación de arquitecturas complejas y con diferentes tecnologías [Pressman, 2009]. Para el desarrollo de la solución se utilizarán los *middleware* orientados a mensajes, conocidos por sus siglas como MOM.

Los Middleware Orientados a Mensajes (MOM)

Los MOM están diseñados para gestionar el intercambio de mensajes entre procesos de forma asíncrona. De esta manera las aplicaciones no se conectan directamente entre ellas, solo se encargan de enviar y recibir mensajes mediante un sistema de colas [Rob, 2006]. Dentro de este tipo de *middleware* existen dos paradigmas de comunicación:

Punto a Punto: Consta de dos nodos de comunicación, uno que recibe y otro que envía. Si el receptor no está disponible, el mensaje se le remite a una cola para que pueda recibirlo en otro momento.

Publicador/suscriptor: Consta de varios nodos de comunicación que pueden ejercer el papel de suscriptor, de publicador o ambos. Ofrece una potente abstracción para multidifusión y comunicación en grupo. Dentro de este paradigma, podemos encontrar cuatro modelos arquitectónicos de comunicación:

- **Arquitectura centralizada con intermediarios**: Todo el tráfico de las entidades pasa por un solo servidor central, que sirve como punto de enlace.
- **Arquitectura centralizada multi-intermediarios:** Cada cola se encuentra almacenada en diferentes servidores conectados entre ellos.
- **Arquitectura descentralizada multi-intermediarios:** El servicio basado en el paradigma publicador/suscriptor distribuye los mensajes internamente entre los servidores.
- **Arquitectura descentralizada sin intermediarios:** No existen servidores, el proceso se realiza en colas a nivel local y los clientes se comunican punto a punto.

Protocolos y normas en *middleware* orientados a mensajes

STOMP

El protocolo de mensajes orientado a texto simple o en flujo (del inglés STOMP) es muy sencillo y fácil de implementar. Se trata de un protocolo de colas de mensajes que se utiliza para la comunicación distribuida entre computadoras. Ofrece un formato de conexión interoperable para que los clientes STOMP puedan comunicarse con cualquier intermediario de mensajes STOMP, proporcionando así, una fácil y generalizada interoperabilidad de mensajería entre muchos lenguajes, plataformas y corredores. Todos los mensajes que se pueden crear, enviar, recibir y procesar son mensajes de texto [STOMP, 2014].

Aunque tiene potencialidades, no es conveniente para la solución propuesta, ya que no soporta la resistencia a fallos. No brindan opciones de configuración referente a los mensajes ni a las colas que los contienen. No admite el enrutamiento de mensajes por patrones.

MQTT

Message Queue Telemetry Transport (MQTT) es un protocolo de conectividad abierto máquina-máquina, que permite enviar datos estilo telemetría, como mensajes. Está diseñado para una mensajería de publicación/suscripción extremadamente simple y ligera. Es ideal para dispositivos con bajas prestaciones y para redes con poco ancho de banda, latencias elevadas y no confiables. Su principio de diseño está basado en lidiar con estas restricciones a la vez que trata de asegurar la fiabilidad y cierto grado de garantía de entrega. Es ampliamente utilizado en la telefonía móvil por su envío eficiente [MQTT, 2014].

El MQTT no es factible para el cumplimiento del objetivo. No proporciona perdurabilidad a los mensajes que se generan. Tampoco posibilita la configuración referente a los mensajes ni a las colas que lo contienen. Su principio de diseño está un poco alejado del ambiente en que se desenvuelve la solución que se propone.

JMS

El servicio de mensajería de java (del inglés JMS) es una API[8] Java de código abierto diseñado por Sun y otras compañías en 1998. Permite a las aplicaciones crear, enviar, recibir y leer mensajes, definiendo un conjunto de interfaces, que posibilitan a los programas escritos en Java comunicarse con otros mediante mensaje [OpenJMS, 2014].

JMS trabaja con los paradigmas de comunicación de los MOM de la siguiente manera:

- **Mensajería punto a punto**: Un productor construye los mensajes y los envía a una cola específica. Luego los clientes suscritos a esa cola pueden extraer los mensajes. Los mensajes permanecerán en la cola hasta que sean consumidos o su tiempo de expiración termine [OpenJMS, 2014].

- **Mensajería publicador/suscriptor**: Un productor construye un mensaje y los direcciona a un nodo intermedio denominado *topic* que lo consume, a partir de aquí el sistema se encarga de distribuir los mensajes publicados a las colas de los suscriptores. Los *topic* solo retienen los mensajes el tiempo necesario para distribuirlos a los suscriptores [OpenJMS, 2014].

[8] Interfaz de aplicación. Es utilizada por un programa para interactuar con las funcionalidades de otro programa o de un equipo físico.

La utilización de JMS no es adecuada para la solución propuesta, ya que está pensado principalmente, para establecer comunicaciones mediante componentes, implementados en el lenguaje de programación Java. Es en ese entorno donde JMS explota sus potencialidades y la implementación de la Plataforma de Desarrollo Integración Continua de Nova difiere en ese sentido.

AMQP

El protocolo avanzado de espera de mensajes (del inglés AMQP) es un estándar abierto para la comunicación mediante un *middleware* orientado a mensajería. Su objetivo principal es alcanzar la interoperabilidad entre diferentes servicios. Proporciona un espacio de colas compartido, accesibles para todas las aplicaciones interesadas en el intercambio de mensajes. Permite tanto el enrutamiento punto a punto como el publicador/suscriptor [Videla, 2012].

El encaminamiento de los mensajes es llevado a cabo por los *brokers* o intermediarios, que redirigen los datos a su destino. Los mensajes tienen una estructura fija con propiedades opcionales relacionadas con la prioridad, el tiempo de vida, el modo de entrega y la perdurabilidad de los datos. Si la aplicación suscriptora no puede procesar los mensajes entrantes, el intermediario los almacena en una cola, garantizando que sean entregados en el mismo orden que llegaron. Las colas también incluyen características asociadas a la persistencia, exclusividad y autoborrado de los mensajes [AMQP.org, 2014].

AMQP es el protocolo que se empleará mediante un MOM para la solución propuesta en la presente investigación, y su elección se basa en las siguientes características esenciales que lo diferencia de otros estándares de *middleware*:

- **Seguridad**: Proporciona una infraestructura para una red con transacciones seguras y de confianzas. Soporta la perdurabilidad de los mensajes independientemente de la conexión de los receptores. La entrega de mensajes es resistente a fallos técnicos.
- **Confiable:** Capaz de eliminar las brechas y retrasos de diferentes plataformas, sistemas y componentes críticos de aplicaciones, tanto dentro como fuera de las empresas. Garantiza la entrega de los mensajes, implementando una semántica que abarca: al menos una vez, a lo sumo una vez y solo una vez, conocido como entrega fiable.
- **Unificado:** Provee un conjunto de patrones básicos de mensajería mediante un único protocolo manejable: mensaje dirigido asíncrono, publicador/suscriptor, solicitud/respuesta, almacenar y reenviar.

1.3.1 Apache ActiveMQ

Apache ActiveMQ es un intermediario de mensajes escrito en java junto con un completo cliente JMS. Es de código abierto y se encuentra bajo la licencia de Apache 2.0. Ofrece soporte para el marco de trabajo Spring de java y para las transacciones. Proporciona alta disponibilidad, rendimiento, escalabilidad, fiabilidad y seguridad para la mensajería empresarial. Está diseñado para comunicarse a través de diferentes protocolos, entre los que destacan algunos como STOMP, AMQP, OpenWire y XMPP. Aunque implementa las especificaciones de JMS, cuenta con APIs para diferentes lenguajes de programación como, C, C++, C#, .NET, Python, PHP y Erlang por solo mencionar los más utilizados [ActiveMQ.org, 2014].

Garantiza la perdurabilidad de los mensajes, mediante la utilización de bases de datos relacionales con buen rendimiento. Las bases de datos pueden ser utilizadas además para realizar copias de seguridad y respaldo de los mensajes, lo cuál no en todos los escenarios resulta beneficioso. En caso de fallos un intermediario ActiveMQ puede ser manejado mediante una configuración maestro/esclavo con el fin de mantener las operaciones en funcionamiento. En el paradigma de comunicación publicador/suscriptor las colas y temas son virtuales, una cola está disponible en todos los nodos, y el enrutamiento se lleva a cabo automáticamente [ActiveMQ.org, 2014].

No es conveniente utilizarlo para la solución propuesta, puesto que las bases de datos relacionales de ActiveMQ para la perdurabilidad de los mensajes, junto a la utilizada por la plataforma, puede ocasionar importantes pérdidas de velocidad.

1.3.2 ZeroMQ

ZeroMQ no es un intermediario en sí, sino más bien una biblioteca de *socket*s que actúa como marco de trabajo concurrente. Sin embargo dada sus características es importante tenerlo en cuenta, cuando hablamos de este tipo de tecnologías. Puede conectar *sockets* punto a punto a través de patrones como *fanout*, *pub-sub*, distribución de tareas y petición-respuesta. Presenta un modelo de entrada-salida que ofrece a las aplicaciones de paso de mensajes un multinúcleo escalable. Cuenta con APIs para un gran número de lenguajes incluyendo los más utilizados como C, C++, Python, C#, java y .NET. Es desarrollado por la corporación iMatix y se encuentra bajo la licencia de código abierto LGPL en su versión [ZeroMq.org, 2014].

A continuación se exponen algunos de sus modelos más importantes:

- **REQ/REP (petición, respuesta):** En el modelo los mensajes son enviados desde un productor a un consumidor. El consumidor realiza alguna acción con el mensaje e informa al productor que lo ha recibido. Los mensajes en ZeroMQ son una cadena de caracteres, por lo que ya no es necesario trabajar a bajo nivel con datos binarios como en otros protocolos.

- **PUB/SUB:** En el modelo los productores envían los mensajes hacia una cola destino. La cola filtra los mensajes de acuerdo a determinados patrones predefinidos por los consumidores y se los entrega. Los consumidores pueden suscribirse a tantas colas como deseen y ZeroMQ se encargará de filtrar los mensajes eficientemente. No controla si los consumidores han recibido el mensaje, sino que asume que así fue, dejando en mano de los desarrolladores la solución del problema.

- **PUSH/PULL:** El modelo consta de un emisor y un número indeterminado de receptores elegidos aleatoriamente para procesar los mensajes antes de entregarlos al consumidor. De esta forma la carga del trabajo es distribuida y los sistemas alcanzan alta escalabilidad.

ZeroMQ no es adecuado para la solución propuesta debido a que simplifica el protocolo TCP/IP[9], pero no deja de ser complejo su uso. No asegura la permanencia de los mensajes en la cola, dado el caso de que los usuarios no puedan recibirlos. Implementar un mecanismo de perdurabilidad que resuelva dicho problema, consumiría un tiempo mayor que el que se dispone para desarrollarla.

1.3.3 RabbitMQ

RabbitMQ es un intermediario para la mensajería de código abierto escrito en Erlang[10]. Es especialmente adecuado para aplicaciones distribuidas y su instalación se ejecuta de forma rápida. Puede ser instalado fácilmente tanto en Linux como en Windows y Mac OS. Utiliza el marco de trabajo de la plataforma abierta de telecomunicaciones (del inglés OTP) para construir sus capacidades de ejecución distribuida y conmutación ante errores [Videla, 2012].

Ofrece alto rendimiento, fiabilidad, escalabilidad y disponibilidad. Incluye la persistencia de los mensajes, así como su acuse de recibo. Permite la agrupación de varios intermediarios RabbitMQ, en una red local, formando un solo agente lógico. Tiene disponible diferentes bibliotecas para la programación en Java y ofrece un conjunto amplio de APIs de lenguajes de programación, entre los que se encuentran C, C++,

[9] Protocolo de Control de Transmisión / Protocolo de Internet (del inglés *Transmission Control Protocol/Internet Protocol*). Es un sistema de protocolos que hace posible la comunicación entre ordenadores que no pertenecen a una misma red.

[10] Lenguaje de programación concurrente y adaptado a las nuevas tecnologías de sistemas multi-core.

C#, .NET, Python, Ruby, Erlang y PHP. Mediante plugin[11] utiliza protocolos como STOMP, MQTT, AMQP 1.0 y brinda una interfaz web atractiva que permite una fácil administración con un conjunto de estadísticas referente al rendimiento y el consumo de recursos. Su núcleo se encuentra totalmente orientado al protocolo estándar abierto AMQP en sus versiones 0-9-1, 0-9 y 0-8. Empresas prestigiosas como Google y Mozilla ha confiado en sus cualidades para desarrollar diferentes proyectos [Videla, 2012].

RabbitMQ es ideal para la solución propuesta y su utilización se fundamenta en:

- La orientación de su núcleo al estándar AMQP, y la interoperabilidad, confiabilidad e integración que le confiere.
- La capacidad que posee para la gestión de las colas y de los mensajes, facilitando la concurrencia de usuarios.
- La garantía que ofrece, en caso de fallos en el sistema, brindando un lugar seguro, para la perdurabilidad de los mensajes, hasta que se restablezcan los servicios, respetando el orden de prioridad.
- La absoluta confianza de que los mensajes han sido recibidos por sus receptores, mediante la funcionalidad de entrega garantizada.

1.4 Metodología de Desarrollo de software

La concepción de un software de calidad en poco tiempo, puede ser todo un reto para los desarrolladores. No aplicar un procedimiento adecuado durante el proceso de desarrollo, puede llevar al fracaso rotundo e inevitable. Es por eso que desde hace algún tiempo se vienen utilizando las metodologías, que imponen un proceso disciplinado sobre el desarrollo de software, con el fin de hacerlo más predecible y eficiente. Dentro del amplio grupo de las metodologías que han surgido con el transcurrir de los años se encuentra Open up, la cual por sus cualidades, será utilizada para el logro del objetivo de la presente investigación.

Open up

Open UP es un marco de trabajo de procesos de desarrollo de software de código abierto. Es un proceso modelo y extensible, dirigido a la gestión y desarrollo de proyectos de software basados en desarrollo iterativo, ágil e incremental; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo. Abarca todo el ciclo de vida de proyectos mediante la ejecución de un conjunto de actividades que se relacionan entre sí, roles y artefactos. Plantea el desarrollo de software en iteraciones, de forma

[11] Es una aplicación que se relaciona con otra para brindarle una nueva función más específica.

que cada una resulta en un incremento. Comprende cuatro fases: Inicio, Elaboración, Construcción y Transición (Ilustración 1), las cuales se detallan a continuación [OpenUP.org, 2014]:

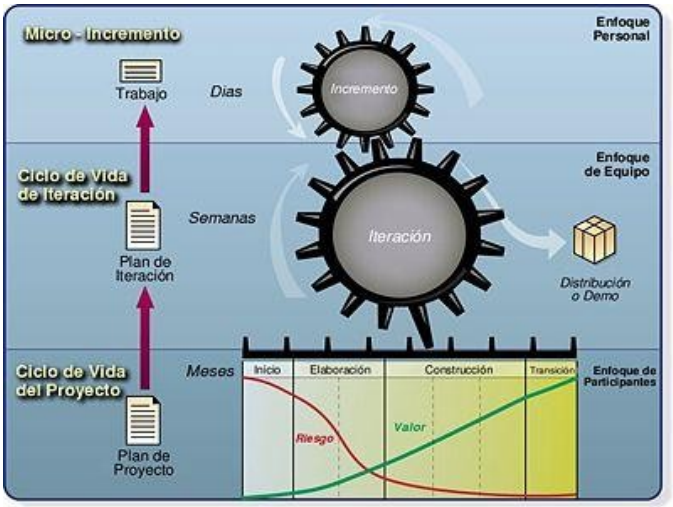

Ilustración 1: Metodología Open UP

1) **Inicio:** Tiene como meta fundamental conocer y entender los objetivos a cumplir durante el proyecto, es en esta fase donde se conocen los requisitos que debe cumplir el producto, se determinan los costos del proyecto y se hace un análisis de factibilidad que muestra las ventajas y desventajas de iniciar el nuevo proyecto de desarrollo.
2) **Elaboración:** Tiene por finalidad obtener una arquitectura lo más robusta posible, tratando por tanto los riesgos significativos para la misma.
3) **Construcción:** El propósito de esta fase es proporcionar una capacidad operacional al sistema, que cumpla con los requisitos especificados, basado en la arquitectura definida.
4) **Transición:** Su objetivo es lograr una versión estable del sistema que pueda ser utilizado por los usuarios, así como su validación para determinar si cumple con las expectativas.

1.5 Herramienta CASE

Las herramientas de software asistidas por computadora (CASE por sus siglas en inglés) son utilizadas para apoyar una o más fases del proceso de desarrollo de software. La calidad de los productos modernos se debe en gran medida a su proceso de desarrollo y las herramientas que lo soportan. El proceso y las herramientas vienen en el mismo paquete: las herramientas son esenciales en el proceso [Jacobson, 1995].

Para el desarrollo de la solución fue seleccionada la herramienta CASE Umbrello, puesto que permite abarcar el ciclo de vida completo del desarrollo de software: análisis y diseño, construcción, prueba y despliegue.

Umbrello UML Modeller

Umbrello UML Modeller es una herramienta para el modelado del proceso de desarrollo de software. Soporta estándares como el Lenguaje Unificado de Modelado, y el intercambio de Metadatos en XML. Permite la ingeniería inversa de los lenguajes C++, Java, Python, Ada, Pascal, IDL y Perl. Así como la generación de código a partir de diagramas para estos y otros lenguajes, como PHP, JavaScript, C#, SQL y Rubi. Es un producto extensible de software libre y de código abierto. La distribución de los modelos puede realizarse exportándolos en los formatos DocBook y XHTML, lo que facilita los proyectos colaborativos. También presenta la funcionalidad de generar PDF y exportar como imagen [Quiñones, 2008].

Los tipos de diagramas que se pueden crear mediante Umbrello en la versión utilizada (2.8.5) son los siguientes:

• Diagrama de casos de uso.

• Diagramas de componentes.

• Diagrama de despliegue.

• Diagrama de entidad-relación.

• Diagrama de clases.

• Diagrama de secuencia.

• Diagrama de estados.

• Diagrama de actividades.

• Diagrama de colaboración.

1.6 JSON

La notación de objetos de JavaScript (JSON por sus siglas en inglés) es un formato de intercambio de datos ligero, que constituye una estructura de datos universal. Es fácil de leer y escribir por parte de los

programadores y muy fácil de parsear y generar en las computadoras. Se caracteriza por un formato de texto que es completamente independiente de los lenguajes de programación y utiliza convecciones que son muy familiares para los programadores en lenguajes como C, C++, C#, Python y JavaScript. Estas propiedades hacen de JSON ideal para el intercambio de datos [JSON.org, 2014]. JSON se utiliza en la solución como estructura de datos para los mensajes que son generados y posteriormente enviados hacia los consumidores.

1.7 Lenguaje de Programación Python 2.7

Python es un lenguaje de programación interpretado creado por Guido van Rossum en el año 1991. Su implementación se encuentra bajo la licencia de código abierto, administrada por la Fundación Python de Software que permite su libre distribución y uso. Su filosofía de diseño enfatiza en la legibilidad del código y su sintaxis es clara y expresiva. Es un lenguaje multi-paradigma, que no fuerza un estilo de programación en particular, sino que permite la programación orientada a objeto, la programación estructurada, así como la programación funcional y la orientada a aspectos. Incluye módulos que proporcionan entradas y salidas de ficheros, llamadas al sistema y hasta interfaces gráficas de usuario (GUI) como TK, GTK, QT, y PythonCard. Python se utiliza como lenguaje de programación interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, al no tener que compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa [Gonzales, 2007].

1.8 Marco de trabajo para aplicaciones web: Django 1.3.1

Django es un marco de trabajo para la web 2.0 de alto nivel y de código abierto escrito en Python. Se trata de una modificación del patrón arquitectónico Modelo-Vista-Controlador (MVC), a la que han llamado Modelo-Plantilla-Vista (MTV por sus siglas en inglés). Su meta fundamental consiste en el desarrollo de sitios complejos conducidos por la base de datos. Utiliza el lenguaje de programación Python para las configuraciones, archivos y modelos de datos. Provee una interfaz administrativa opcional generada dinámicamente y configurada mediante modelos de administración [Holovaty, 2007].

Algunas de sus características más relevantes son [Holovaty, 2007]:

- **Mapeador Objeto-Relacional**: Permite definir los modelos de datos mediante código Python, y permite acceder a ellos mediante una API para las diferentes bases de datos soportadas.
- **Diseño elegante de URL:** Permite crear URLs sencillas y claras mediante expresiones regulares.

- **Sistema de plantillas:** Posee un lenguaje de plantilla robusto y eficiente, que separa el diseño, contenido y código Python de una manera elegante.

- **Internacionalización:** Presenta soporte total para crear aplicaciones multi-lenguajes.

Django unido al lenguaje de programación Python constituyen las tecnologías base en la Plataforma de Desarrollo e Integración Continua de Nova. De ahí que jueguen un papel relevante en la implementación de la solución del lado del servidor.

1.9 PyQt: Qt 4.8.1 para Python

PyQt es una adaptación de la biblioteca gráfica Qt, para el lenguaje de programación Python. Distingue por su sencillez, y por poseer un número importante de herramientas que gestionen su manipulación. Su carácter multiplataforma, facilita su despliegue en los principales sistemas operativos. El diseño se basa en lograr mucho, con poco código. La facilidad de Python unido a la excelencia de Qt, hace más agradable la programación visual. Cuenta con una amplia documentación, y a partir de su versión 4 utiliza *widgets*[12] nativos para las distintas plataformas, por lo que ya no necesita emular el aspecto de la plataforma en la que corre, lo cual es una mejora de rendimiento y calidad visual. Con PyQt se pueden realizar interfaces sencillas y complejas sin muchos contratiempos [PyQt4.org, 20014].

Qt Designer 4.8.1

Qt Designer es una herramienta para el desarrollo de formularios y presentaciones gráficas de aplicaciones. Permite acelerar la elaboración de interfaces de alto rendimiento de forma fácil, generando el código fuente para que los desarrolladores puedan adaptarlo a sus necesidades. Es multiplataforma y provee poderosas opciones como la previa visualización de la interfaz, soporte para *widgets* y un editor de propiedades con gran variedad de opciones [PyQt4.org, 20014]. A raíz de las características que brindan estas poderosas herramientas, se utilizan como elementos fundamentales en la creación de los componentes e interfaces de usuarios en la solución propuesta del lado de los clientes.

1.10 IDE: Eclipse Indigo con el plugin PyDev 2.8.2

Eclipse es un entorno de desarrollo integrado (IDE por sus siglas en inglés) escrito en java. Es extensible mediante plugin, por lo que puede ser utilizado para desarrollar aplicaciones en diferentes lenguajes de programación como Ada, C, C++, COBOL, Fortran, Haskell, Perl, PHP, Python, R, Ruby, Scala, Clojure, Groovy, y Scheme. Presenta funcionalidades para administrar el espacio de trabajo, compilar, ejecutar y

[12] Aplicación o programa pequeño que puede ser incrustado en otras aplicaciones, ejemplo: relojes, agendas, calculadoras.

depurar aplicaciones. Permite compartir elementos y ofrece control de versiones sobre el código fuente. La web oficial de Eclipse lo define como “una plataforma (IDE), abierta para todo y para nada en particular”. PyDev es un plugin elaborado por terceras personas para Eclipse [PyDev.org, 20014]. Es usado para programar en Python y soporta numerosas funcionalidades, entre las que se encuentran:

- Soporte para CPython, Jython e IronPython.
- Realzado de sintaxis con reconocimiento de sintaxis para Python 2.x y Python 3.x.
- Integración con el marco de trabajo Django.
- Completamiento de código con auto *import*.
- Completamiento de palabras reservadas mientras se escribe.
- Análisis de código (con solucionado rápido o “*quick fix*” – Ctrl + 1).
- Acceso rápido a la definición de símbolos con la funcionalidad “*Go to definition*”.
- Refactorización.
- Marcado de ocurrencias.
- Depurador remoto.
- Explorador de *tokens*.
- Consola de depuración.
- Consola interactiva.
- Integración con *Unittest*.
- Errores de analizador sintáctico en tiempo de edición.
- Selección de preferencia: tabuladores o espacios.
- Incremento y decremento de sangría inteligente.
- Comentado y descomentado rápido de bloques.
- Enrollado de código (*code folding*).

Debido a las potencialidades que brindan para el desarrollo de aplicaciones programadas en Python y su integración con el marco de desarrollo web Django, resultan adecuadas para la implementación de la solución.

Conclusiones parciales

- El estudio de las principales formas de comunicación presentes en diferentes sistemas de notificaciones, de las redes sociales Facebook y Twitter, el blog Cubadebate, y la plataforma Launchpad. Unido a su amplio uso por parte de los usuarios para mantenerse informado, arrojó la necesidad de que la Plataforma de Desarrollo e Integración Continua de Nova, incluya un sistema de notificaciones para la comunicación con los involucrados en los procesos que en ella se desarrollan, capaz de soportar la inclusión de diferentes vías de comunicación, accesibles tanto desde sus computadoras como por la telefonía móvil.

- El análisis de los diferentes protocolos y *middleware* orientados a mensajes permitió establecer al protocolo AMQP y al *middleware* RabbitMQ que lo implementa de forma nativa, como, herramientas ideales para llevar a cabo la solución que se propone.

- A partir de un grupo de diferentes tecnologías utilizadas por los integrantes del DSO del centro CESOL para la concepción de su plataforma, y del análisis al problema planteado, se pudo determinar las herramientas que conforman el ambiente de desarrollo.

Capítulo 2: Desarrollo de la Solución

El presente capítulo, ilustra el proceso de desarrollo de la solución al problema planteado mediante el transcurso de las diferentes fases de la metodología Open up, y un grupo de artefactos ingenieriles que posibilitan la obtención de un producto de calidad.

2.1 Propuesta de Solución

La solución propuesta consiste en un sistema de notificaciones para la Plataforma de Desarrollo e Integración Continua de Nova basado en el modelo cliente-servidor y el patrón Broker. Su arquitectura genérica permite al administrador elaborar notificaciones referentes a cada uno de los componentes de la plataforma a medida que se vayan incluyendo. Es capaz de adaptarse fácilmente a la incorporación de nuevos módulos y sus respectivas notificaciones, permitiendo a su vez la resistencia de la información a fallos técnicos tanto en el servidor como en el cliente, mediante las opciones de perdurabilidad brindadas por Rabbitmq y AMQP. El procedimiento asíncrono que emplea, posibilita que la plataforma pueda concentrar su poder de cómputo en las tareas más críticas del proceso, sin necesidad de esperar respuestas del envío de los mensajes, pero con un grado de confianza elevado referente a su recepción. En cuanto al cliente, se propone una aplicación de escritorio, que se incrustan en la barra de notificación de los sistemas operativos brindando información a los usuarios en todo momento. La aplicación cliente utiliza el paradigma de comunicación publicador/suscriptor, permitiendo a los usuarios elegir sobre qué tema desean ser notificados. Como medidas de seguridad, solo los usuarios registrados en la plataforma pueden acceder a este servicio, y la información viaja por un canal cifrado.

2.2 Modelo de dominio

Para una mayor comprensión de la propuesta de solución, en la ilustración 2 se ha representado el modelo de dominio de las principales clases conceptuales.

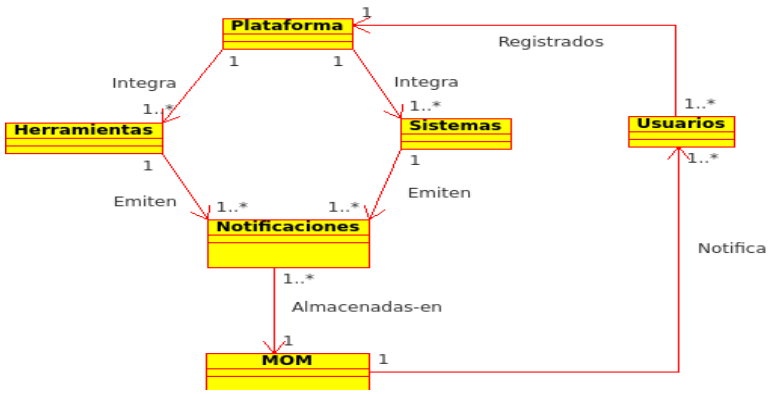

Ilustración 2: Modelo de dominio de la solución propuesta.

2.3 Requisitos del sistema

Un requisito es una descripción detallada, en las cuales el cliente describe brevemente las características y cualidades que un sistema informático debe tener [Jacobson, 1999]. Para una mejor especificación se dividen en dos grupos, requisitos funcionales y no funcionales.

2.3.1 Requisitos funcionales

1. Adicionar conexión con el servidor middleware.
2. Modificar conexión con el servidor middleware.
3. Eliminar conexión con el servidor middleware.
4. Adicionar una notificación.
5. Modificar una notificación.
6. Eliminar notificación.
7. Capturar el disparador.
8. Generar mensaje asociado al disparador.
9. Enviar mensaje hacia el servidor middleware.
10. Solicitar nombre de usuario y contraseña.
11. Verificar el usuario y contraseña.
12. Listar opciones de notificación.
13. Permitir la selección de las opciones de notificación.
14. Suscribir a las notificaciones.
15. Guardar configuraciones.
16. Cargar configuraciones.
17. Modificar configuraciones.
18. Mostrar la notificación recibida.
19. Actualizar lista de notificaciones.
20. Actualizar conexión con el servidor middleware.

2.3.2 Requisitos no funcionales

Usabilidad

1. Los usuarios finales del sistema de notificación, son desarrolladores de software.
2. El sistema de notificación posee dos formas de funcionamiento, como cliente y como servidor.
3. El sistema de notificación se emplea para el envío y recepción de mensajes/notificaciones desde el servidor hacia los clientes.

Confiabilidad

1. El sistema debe ser capaz de reanudar las operaciones si el funcionamiento del servidor se ve afectado por la ausencia del fluido eléctrico.
2. El sistema debe ser capaz de reanudar las operaciones si el funcionamiento de las computadoras clientes, se ve afectado por la ausencia de fluido eléctrico.
3. El sistema no debe tener pérdida de mensajes si el funcionamiento del servidor se ve afectado por la ausencia del fluido eléctrico.
4. El sistema no debe tener pérdida de mensajes si el funcionamiento del servidor se ve afectado por la ausencia de conexión en la red.
5. El sistema no debe tener pérdida de mensajes si el funcionamiento de las computadoras clientes se ve afectado por la ausencia de conexión en la red.
6. El sistema no debe tener pérdida de mensajes si el funcionamiento de las computadoras clientes se ve afectado por la ausencia del fluido eléctrico.

Eficiencia

1. Garantizar que el sistema soporte una cantidad escalable de peticiones, proporcionando respuestas efectivas y rápidas.

Restricciones de diseño

1. Para la implementación del sistema se utilizará el lenguaje de programación Python.
2. La comunicación se establecerá mediante el middleware orientado a mensajes RabbitMQ y el protocolo AMQP.
3. Para la implementación del sistema del lado del servidor se utilizará el marco de trabajo para el desarrollo web Django.
4. Para la codificación del lado del servidor y en los clientes de escritorio se utilizará el IDE eclipse y la biblioteca PyDev.
5. Para el modelado de la aplicación se utilizará la herramienta Umbrello.
6. La arquitectura de la aplicación debe responder a los patrones arquitectónicos cliente-servidor y Broker.
7. Para el desarrollo de las interfaces de usuarios de los clientes de escritorio se utilizará la biblioteca PyQt y la aplicación Qt Designer.

8. Los mensajes viajarán por el canal, encapsulados en un JSON.

Seguridad

1. Garantizar el tratamiento de excepciones.
2. Garantizar el control de acceso al sistema.
3. La información debe viajar por un canal cifrado
4. La aplicación cliente guardará las credenciales (usuario y contraseña) de los usuarios en el anillo de llaves de Gnome.

Requisitos de Licencia

1. Se implementará bajo la licencia GPL.

2.4 Actores del Sistema

Actor	Descripción
Administrador	Es el encargado de monitorizar todo las funcionalidades de la aplicación en el servidor, gestiona las notificaciones y las conexiones con el *middleware* de mensajería RabbitMQ.
Usuarios	Representa a una persona que puede autenticarse en la aplicación cliente, y utilizar sus funcionalidades.

2.5 Casos de usos del sistema

Los casos de uso de un sistema, describen un conjunto de actividades desde el punto de vista de los actores, produciendo un resultado concreto y tangible. Son descriptores de las interacciones que se producen entre los usuarios y el sistema. Los casos de usos presentes en la solución son:

1. Gestionar conexión.
2. Gestionar notificaciones.
3. Enviar notificaciones.
4. Autenticar usuarios.
5. Suscribir usuarios.
6. Mostrar notificaciones.
7. Actualizar.

2.6 Diagrama de casos de usos

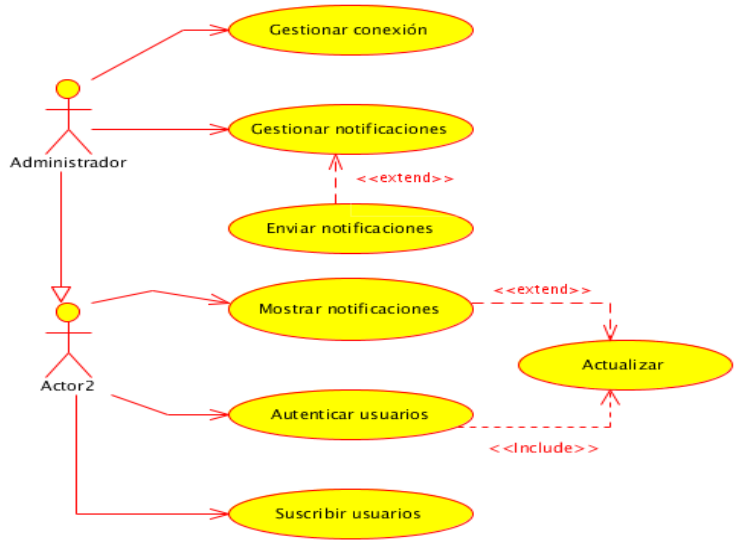

Ilustración 3: Diagrama de casos de uso de la solución propuesta.

2.7 Especificación de los casos de usos críticos

Caso de uso: Gestionar conexión

Objetivo	Adicionar, modificar y eliminar la conexión con el *middleware* RabbitMQ
Actores	Administrador.
Resumen	Adiciona, modifica y elimina la dirección ip, tipo de comunicación y estado que posee el servidor donde se encuentra el *middleware* RabbitMQ.
Complejidad	Baja
Prioridad	Crítico
Precondiciones	El Administrador debe estar autenticado en la Plataforma de Desarrollo e Integración Continua de Nova.
Postcondiciones	
Flujo de eventos	
Flujo básico <Gestionar Conexión>	

	Actor	**Sistema**
1	Accede al formulario adicionar conexión.	Permite realizar varias acciones con una conexión: • Adicionar una nueva conexión. Ver Sección 1: “Adicionar conexión”. • Modificar una conexión. Ver Sección 2: “Modificar conexión” • Eliminar una conexión. Ver Sección 3: “Eliminar conexión”

2		Termina el caso de uso.
Sección 1: “Adicionar conexión”		
Flujo básico <Adicionar>		
	Actor	**Sistema**
1	Selecciona la opción adicionar conexión.	Se muestra el formulario para adicionar una nueva conexión.
2	Introduce la dirección ip y estado del servidor del *middleware* RabbitMQ	Se validan los datos introducidos.
3		Adiciona una nueva conexión estableciendo el puerto 5631 por defecto.
4		Muestra un mensaje de confirmación.
5		Termina el caso de uso.
Flujos alternos		
2.a <Los datos son incorrectos>		
	Actor	**Sistema**
1		Muestra un mensaje de error indicando el campo en que se produjo y el motivo.
Flujo básico <Modificar conexión>		
	Actor	**Sistema**
1	Selecciona la conexión a modificar.	Muestra el formulario para modificar conexión.
2	Modifica la dirección ip el tipo de comunicación o estado del servidor del *middleware* RabbitMQ.	Se validan los datos introducidos.
3		Cambia la conexión y establece el puerto 5631 por defecto.
4		Muestra un mensaje de confirmación.
5		Termina el caso de uso.
Flujos alternos		
2.a<Los datos son incorrectos>		
	Actor	**Sistema**
		Muestra un mensaje de error indicando el campo en que se produjo y el motivo.
Sección 3: “Eliminar conexión”		
Flujo básico <Eliminar>		
	Actor	**Sistema**
1	Selecciona la conexión a eliminar.	Elimina la conexión seleccionada.
2		Muestra un mensaje de confirmación.
3		Termina el caso de uso.
Requisitos no funcionales	Los formularios se muestran mediante la interfaz de administración de Django.	
Prototipos	Ver Anexo 2	

Caso de uso: Gestionar notificaciones

Objetivo	Adicionar, modificar y eliminar las notificaciones.
Actores	Administrador.
Resumen	Adiciona, modifica y elimina las notificaciones que serán enviadas al middleware RabbitMQ.
Complejidad	Media.
Prioridad	Crítico.
Precondiciones	El Administrador debe estar autenticado en la Plataforma de Desarrollo e

	Integración Continua de Nova.	
Postcondiciones		
Flujo de eventos		
Flujo básico <Gestionar Notificaciones>		
	Actor	**Sistema**
1	Accede al formulario de notificaciones.	Permite realizar varias acciones con una notificación: • Adicionar una nueva notificación. Ver Sección 1: “Adicionar notificación”. • Modificar una notificación. Ver Sección 2: “Modificar notificación”. • Eliminar una notificación. Ver Sección 3: “Eliminar notificación”.
2		Termina el caso de uso.
Sección 1: “Adicionar notificación”		
Flujo básico <Adicionar>		
	Actor	**Sistema**
1	Selecciona la opción adicionar notificación.	Muestra el formulario para adicionar una notificación.
2	Introduce el título, cuerpo y disparador, de la notificación así como los datos referentes al modelo del componente al que hace referencia.	Valida los datos introducidos.
3		Adiciona una nueva notificación.
4		Muestra un mensaje de confirmación.
5		Termina el caso de uso.
Flujos alternos		
2a. <Los datos son incorrectos>		
	Actor	**Sistema**
1		Muestra un mensaje de error indicando el campo en que se produjo y el motivo.
Sección 2: “Modificar notificación”		
Flujo básico <Modificar>		
	Actor	**Sistema**
1	Selecciona la conexión a modificar.	Muestra el formulario para modificar notificación.
2	Modifica el título, cuerpo y disparador, de la notificación así como los datos referentes al modelo del componente al que hace referencia.	Valida los datos introducidos.
3		Cambia los valores de la notificación.
4		Muestra un mensaje de confirmación.
5		Termina el caso de uso.
Flujos alternos		
2.a <Los datos son incorrectos>		
	Actor	**Sistema**
1		Muestra un mensaje de error indicando el campo en que se produjo y el motivo.
Sección 3: “Eliminar notificación”		
Flujo básico <Eliminar>		
	Actor	**Sistema**
1	Selecciona la notificación a eliminar.	Elimina la notificación seleccionada.

		Muestra un mensaje de confirmación.
		Termina el caso de uso.
Relaciones	**CU Extendidos**	Enviar notificaciones: Paso 3 del Flujo Básico.
Requisitos no funcionales	Los formularios se muestran mediante la interfaz de administración de Django.	
Prototipos	Ver Anexo 2	

Caso de uso: Autenticar usuarios

Objetivo	Autenticar a los usuarios.	
Actores	Usuarios.	
Resumen	El caso de uso se inicia cuando el actor introduce el URI de la plataforma y su usuario y contraseña, para comprobar que sean correctas y poder acceder a las funcionalidades de las aplicaciones clientes.	
Complejidad	Media.	
Prioridad	Crítico.	
Precondiciones	Debe ser usuario de la Plataforma de Desarrollo e Integración Continua de Nova.	
Postcondiciones	El usuario puede acceder a las funcionalidades de la aplicación cliente y no necesita volverse autenticar hasta que sus credenciales cambien en la plataforma, o sean eliminadas del anillo de llaves de Gnome de su ordenador.	
Flujo de eventos		
Flujo básico <Autenticar usuarios>		
	Actor	**Sistema**
1	Introduce el URI de la plataforma y su usuario y contraseña.	Envía usuario y contraseña por petición https a la plataforma de Nova.
2		Verifica que el usuario y la contraseña sean correctos.
3		Almacena el usuario y la contraseña en el anillo de llaves de Gnome.
4		Actualiza la lista de notificaciones y la conexión del servidor middleware en caso de que existan cambios. Ver caso de uso Actualizar.
5		Muestra la ventana principal con todas las funcionalidades.
6		Termina el caso de uso.
Flujos alternos		
1.<El usuario o la contraseña son incorrectos>		
	Actor	**Sistema**
		Muestra un mensaje de error informando que las credenciales presentadas no son correctas.
Relaciones	**CU Incluidos**	CU Actualizar.
Requisitos no funcionales	Solo pueden acceder los usuarios que introduzcan sus credenciales (usuario y contraseña) correctamente.	
Prototipos	Ver Anexo 3	

Las especificaciones del resto de los casos de usos, se pueden encontrar en las Tablas al final del documento.

2.8 Arquitectura

2.8.1 Patrones de arquitectura

Los patrones arquitectónicos, tienen como objetivo fundamental asegurar que los sistemas informáticos cumplan con los objetivos de las cualidades de software mantenibilidad, escalabilidad, reutilización y disponibilidad. A continuación se presentan los patrones más relevantes en la implementación de la solución.

Modelo-Vista-Controlador

La utilización de este patrón, se encuentra asociado a la utilización del marco de trabajo para el desarrollo web Django que lo implementa. Es evidenciado en la solución mediante el uso de modelos para guardar la información referente a las conexiones con el *middleware* RabbitMQ y las notificaciones generadas por el sistema. Los modelos son manejados por la clase controladora, que en Django se le conoce como vista y se muestra su información mediante plantillas que constituyen la vista en el patrón original.

Broker

Este patrón de arquitectura se utiliza fundamentalmente para organizar sistemas distribuidos con componentes débilmente acoplados, que interactúan entre sí, invocando servicios remotos. Constituye la base de la arquitectura más general de la solución propuesta y brinda una alta portabilidad y reusabilidad. Sus componentes, así como la manera en que se producen las interacciones en la solución propuesta, se pueden observar en la ilustración 4.

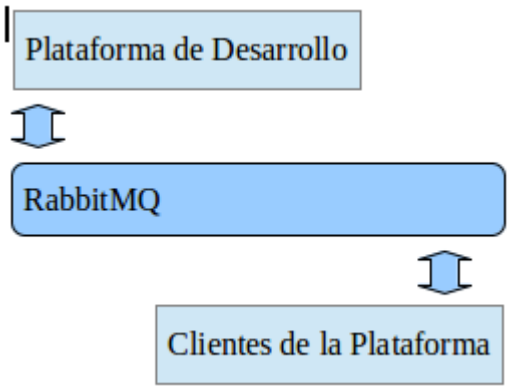

Ilustración 4: Patrón Broker

2.8.2 Patrones de diseño

Los patrones de diseños son soluciones simples y elegantes a problemas específicos de la programación orientada a objetos, donde su eficacia ya ha sido probada. Para el diseño del Sistema de Notificación de la Plataforma de Desarrollo e Integración Continua de Nova, se aplicaron las bases e ideas que proporcionan los patrones GRAPS, los cuales describen los principios fundamentales de asignación de responsabilidades a los objetos o clases.

Patrón Experto: Establece que la responsabilidad de realizar una determinada labor es de la clase que contiene todos los datos necesarios para ello. En la ilustración 5 se evidencia su presencia en el marco de la solución.

Ilustración 5: Ejemplo de la utilización del patrón experto.

Patrón Creador: Guía la asignación de responsabilidades relacionadas con la creación de objetos, estableciendo que una instancia de un objeto la tiene que crear la clase que cuenta con la información necesaria para ello. En la Ilustración 6 se evidencia su presencia en el marco de la solución.

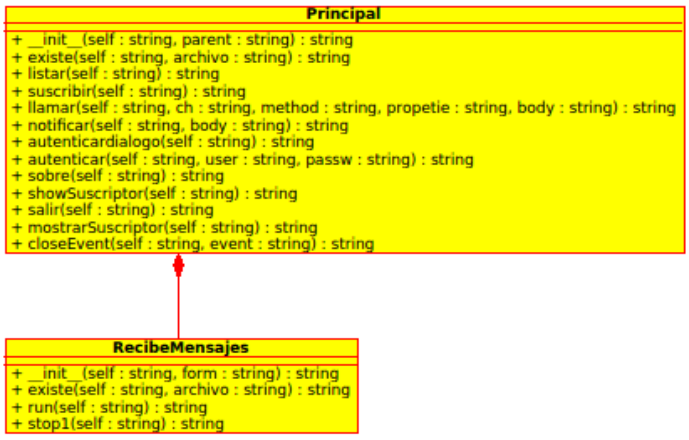

Ilustración 6: Ejemplo de utilización del patrón creador.

Patrón bajo acoplamiento: Propone el diseño de clases con pocas dependencias, lo que reduce el impacto del cambio y facilita la reutilización en otros sistemas. En la ilustración 7 se evidencia su presencia en el marco de la solución mediante la clase mensajero, la cual posee una arquitectura genérica sin dependencias, que le permite ser reutilizada fácilmente por otros modelos y sistemas.

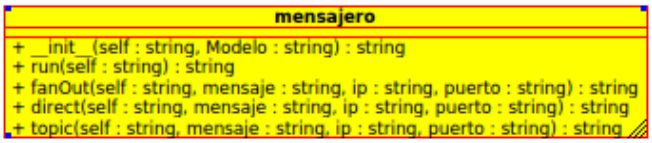

Ilustración 7: Ejemplo de la utilización del patrón bajo acoplamiento.

Patrón Alta Cohesión: Detalla cómo cada clase debe realizar una labor única dentro del sistema, teniendo responsabilidades moderadas en un área funcional y colaborando con las otras para llevar a cabo las tareas. La responsabilidad de este patrón es evitar que se asignen demasiadas responsabilidades a las clases. En la solución se evidencia mediante el desempeño de métodos con funcionalidades únicas y altamente relacionadas. En la ilustración 8 se observa su presencia en el marco de la solución.

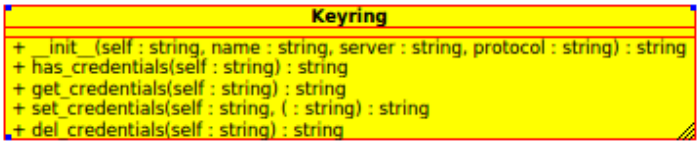

Ilustración 8: Ejemplo de la utilización del patrón alta cohesión

Patrón Controlador: Describe la asignación de la responsabilidad de controlar el flujo de eventos del sistema a clases más específicas. En la solución se evidencia mediante la clase “ Principal “ encargada de la lógica del negocio del lado del cliente.

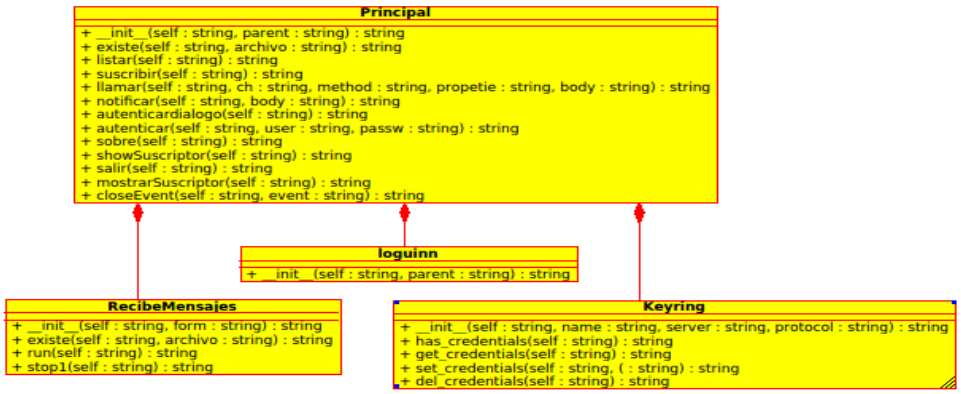

Ilustración 9: Ejemplo de la utilización del patrón controlador.

2.9 Diagramas de clases del diseño

Para una mayor comprensión de la realización física de los casos de usos, tanto del lado del servidor como por parte de la aplicación cliente, se muestran a continuación los diagramas de clases del diseño para ambas implementaciones.

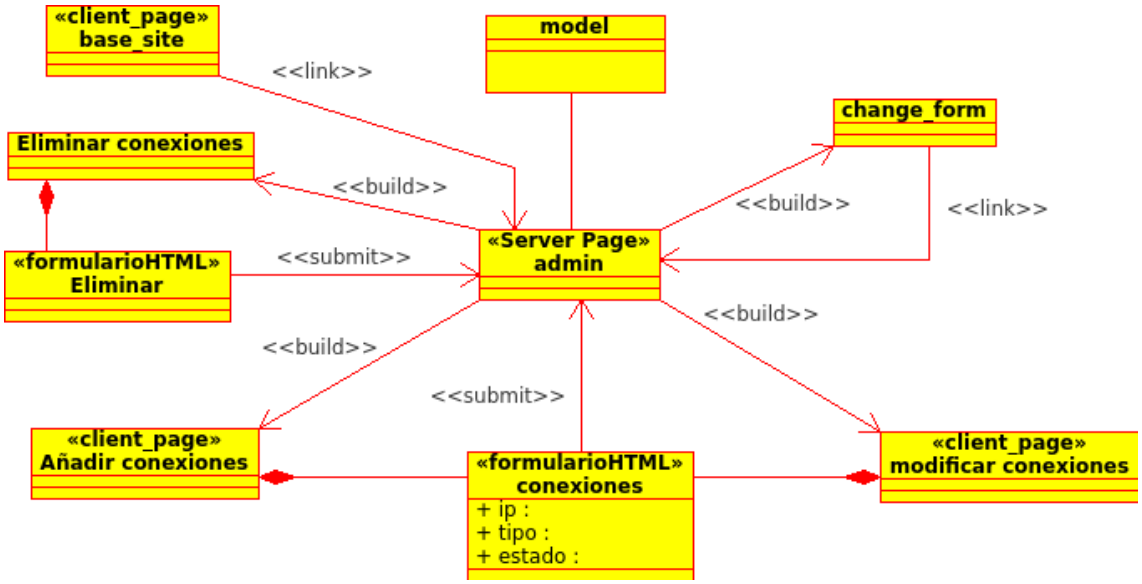

Ilustración 10: diagrama de clases del diseño CU Gestionar conexión

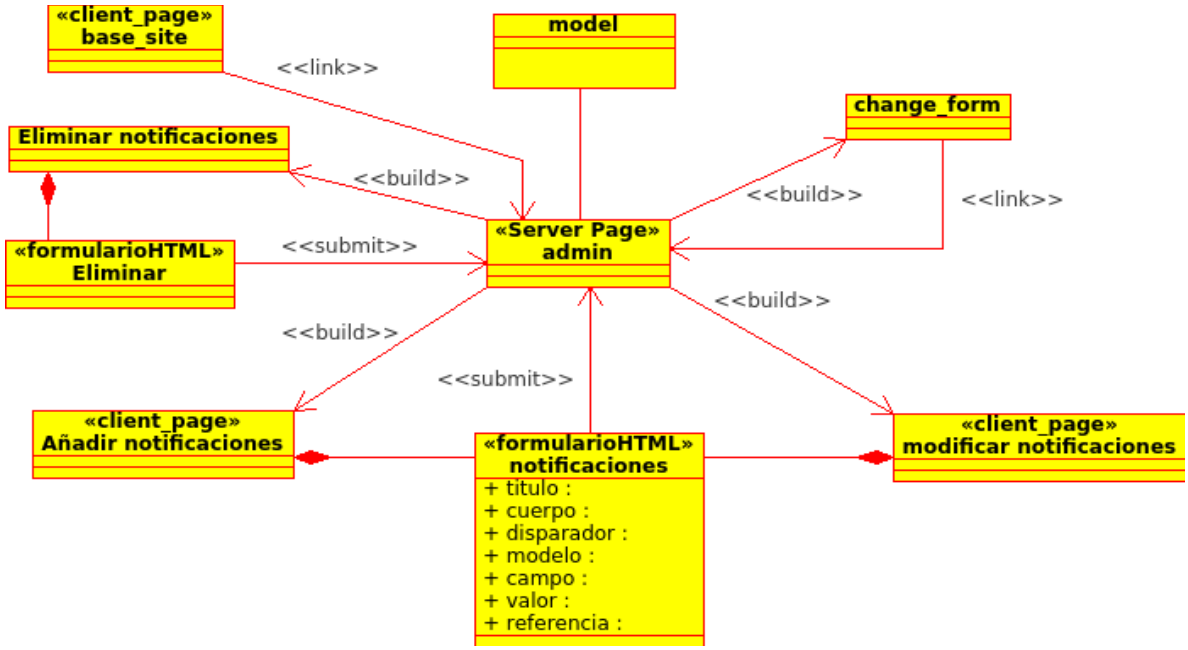

Ilustración 11: Diagrama de clases del diseño del CU Gestionar Notificaciones

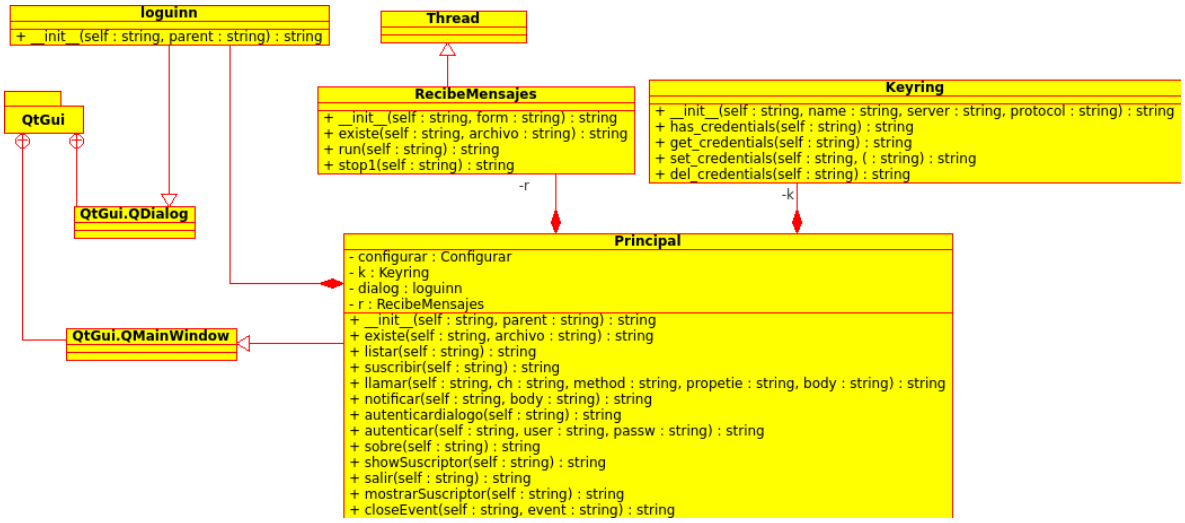

Ilustración 12: Diagrama de clases del cliente de notificaciones.

admin: Es la controladora de toda la lógica del negocio del lado del servidor. Tiene la responsabilidad de gestionar los modelos contenidos en el model.py del marco de trabajo para el desarrollo web Django y construir las vistas con las que interactúa el administrador para gestionar las conexiones y notificaciones.

model: En este módulo se encuentran todas las clases entidades necesarias para la persistencia de la información en la base de datos de la plataforma de desarrollo.

change_form: Constituye la vista principal o plantilla html como es conocida en el ambiente Django, que contiene el formulario principal para la gestión de las conexiones y notificaciones por parte del administrador.

Principal: Clase que constituye la interfaz principal del cliente de notificaciones. Es la encargada de controlar todo el negocio, suscribir a los usuarios y mostrar las notificaciones.

Loguinn: Clase interfaz encargada de autenticar a los usuarios con la Plataforma de Desarrollo e Integración Continua de Nova.

RecibeMensajes: Clase responsable de interactuar con el middleware RabbitMQ. Constituye un proceso ligero que se ejecuta paralelo al hilo principal.

Keyring: Clase con la responsabilidad de introducir y obtener las credenciales de los usuarios en el anillo de llaves de Gnome. Posibilitando así que una vez autenticado, no tengan que volver a introducirlas a menos que cambien sus credenciales en la Plataforma.

2.10 Diagramas de Interacción

Los diagramas de interacción son utilizados para mostrar la interacción entre objetos, modelando así los

aspectos dinámicos del sistema. La solución propuesta define los diagramas de secuencias para cada caso de uso.

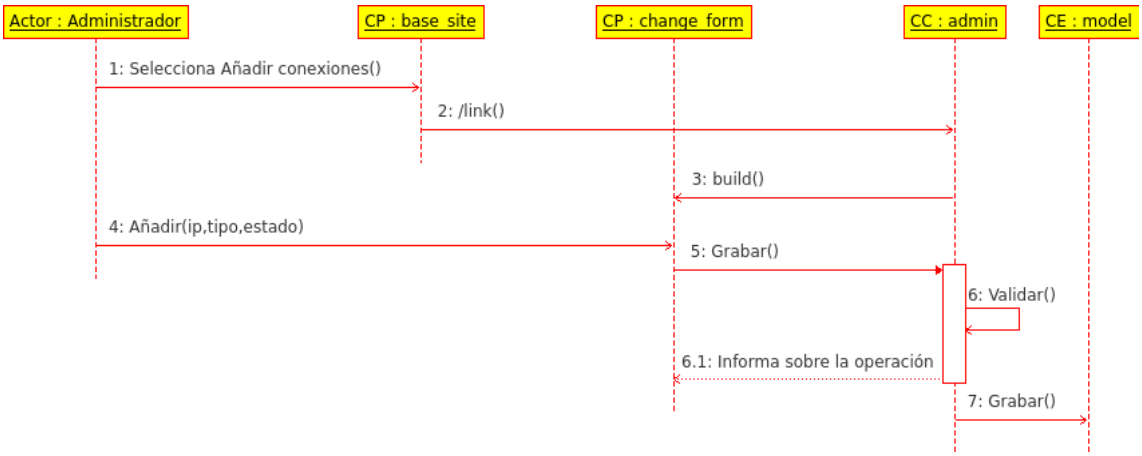

Ilustración 13: Diagrama de secuencia CU Gestionar conexión sección Adicionar conexión.

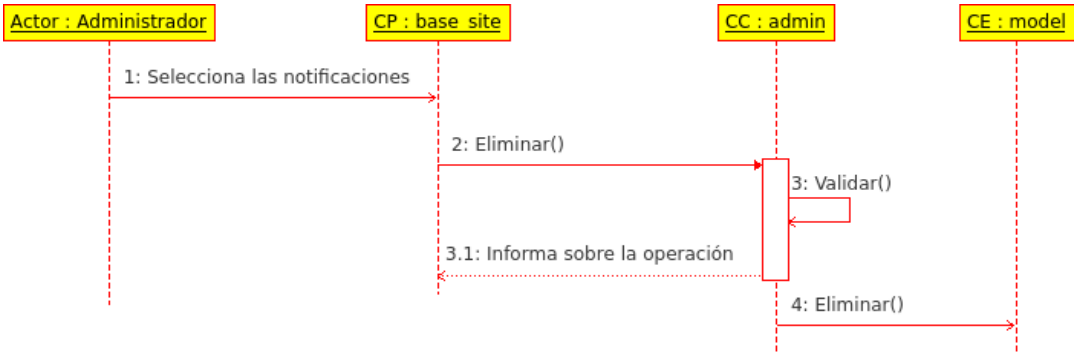

Ilustración 14: Diagrama de secuencia CU Gestionar notificaciones sección Eliminar notificación.

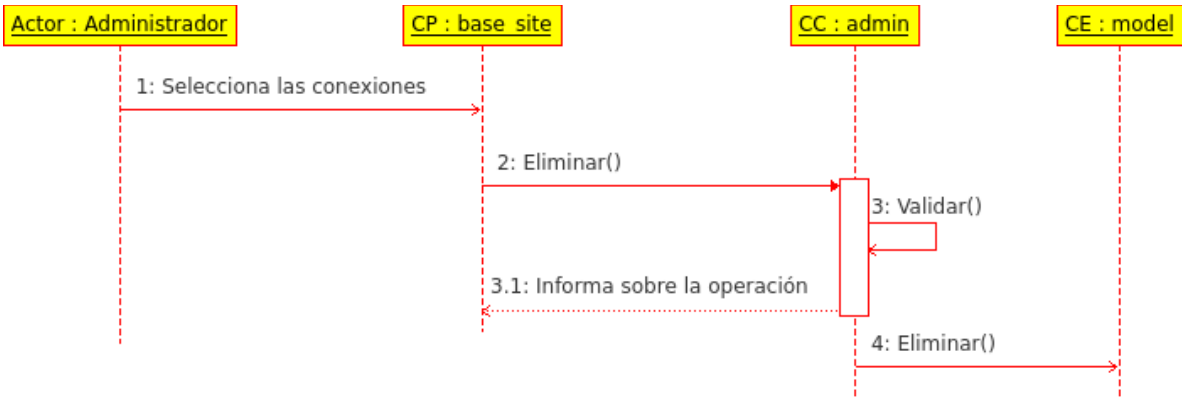

Ilustración 15: Diagrama de secuencia CU Gestionar conexión sección Eliminar conexión.

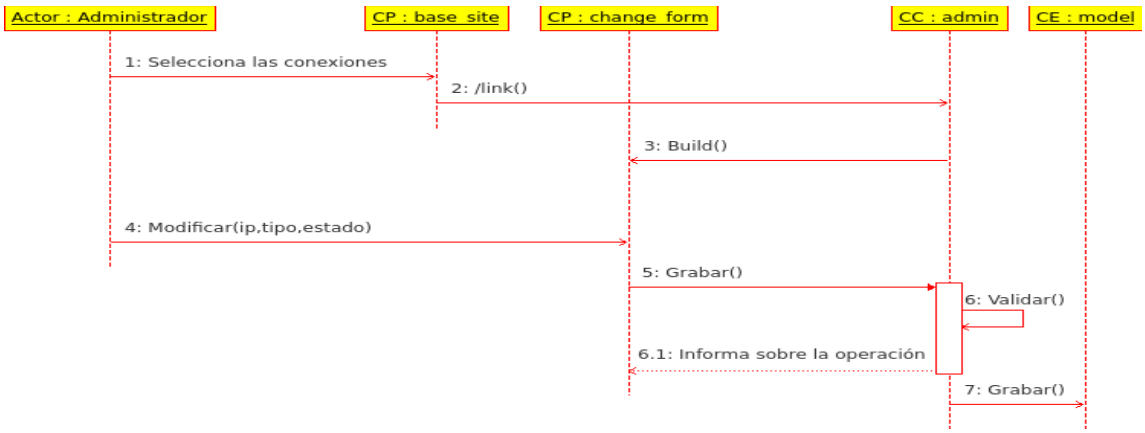

Ilustración 16: Diagrama de secuencia CU Gestionar conexión sección Modificar conexión.

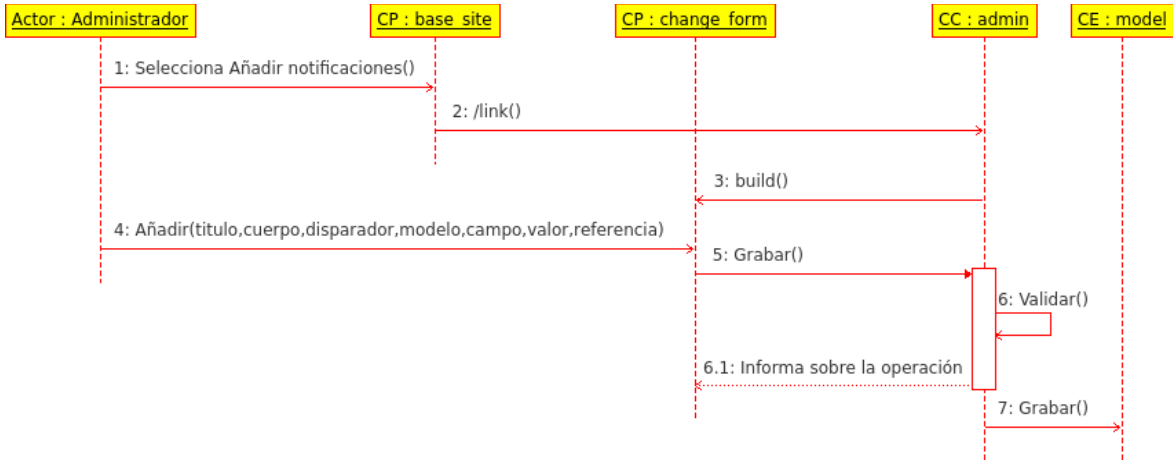

Ilustración 17: Diagrama de secuencia CU Gestionar notificaciones sección Añadir notificación.

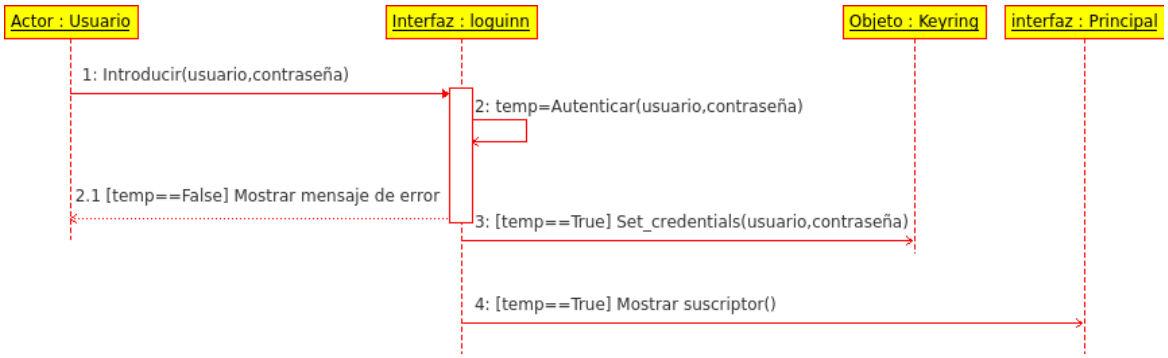

Ilustración 18: Diagrama de secuencia CU Autenticar usuario.

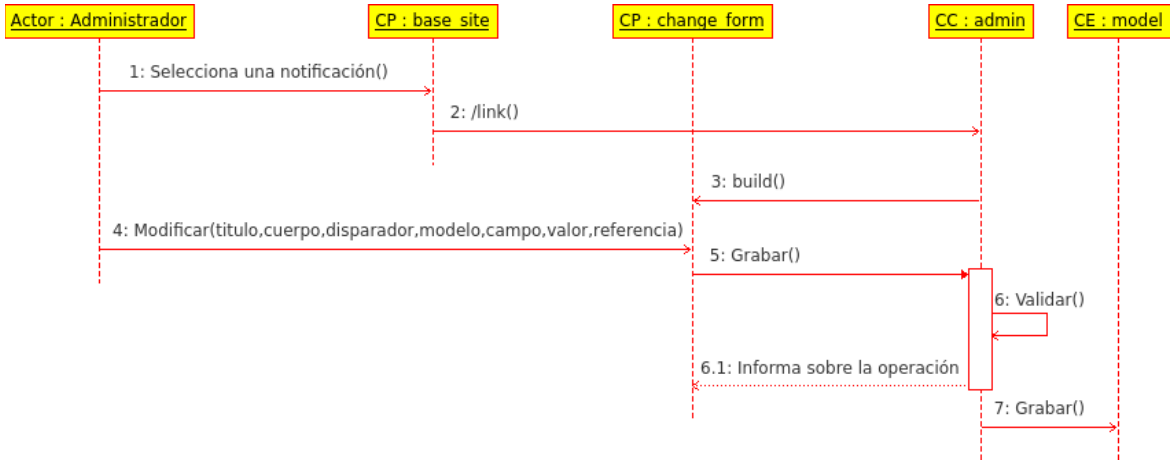

Ilustración 19: Diagrama de secuencia CU Gestionar notificaciones sección Modificar notificación.

El resto de los diagramas de secuencias se pueden encontrar en el Anexo 1.

2.11 Diagrama de despliegue

Para representar las interconexiones entre los nodos físicos de la solución propuesta y el software que los componen se muestra el siguiente diagrama de despliegue.

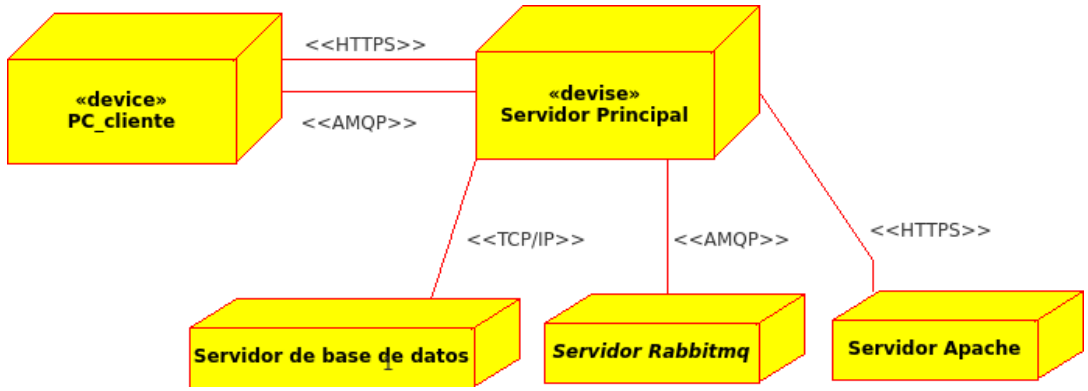

Ilustración 20: Diagrama de despliegue de la solución propuesta.

2.12 Diagrama de componentes

A continuación se muestra el modelo de implementación en términos de subsistemas mediante el diagrama de componentes.

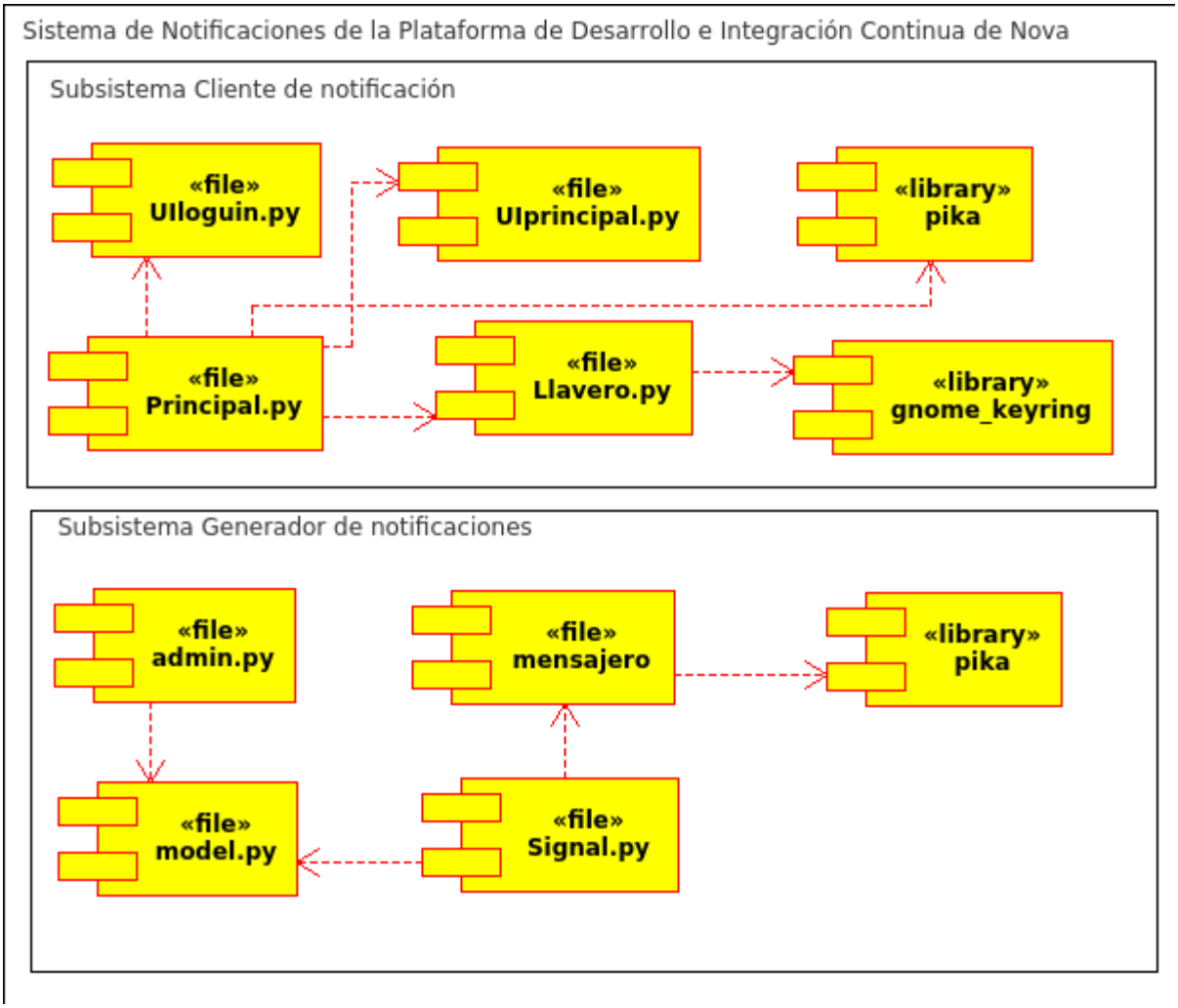

Ilustración 21: Diagrama de componentes de la solución.

2.13 Estándares de codificación

Para garantizar la legibilidad y comprensión del código de la solución propuesta por parte de los programadores, en acciones de mantenimientos futuras. Se adoptó el estándar de codificación PEP-8 [Python.org, 2013]. Asegurando que al añadir nuevas funcionalidades, modificar las existentes, depurar errores o mejorar el rendimiento, pueda realizarse en cortos períodos de tiempo y manteniendo una alta calidad.

Conclusiones parciales

- Se obtuvo una arquitectura robusta, genérica y extensible, que permite al administrador de la Plataforma de Desarrollo e Integración Continua de Nova incorporar notificaciones personalizadas, así como nuevas vías de comunicación con alta escalabilidad.
- Se logró proporcionar una capacidad operacional al sistema, que cumple con los requisitos especificados y se encuentra listo para pasar a la etapa de pruebas.
- Los principales artefactos que rigen la metodología Open up para las fases de elaboración y construcción, permitieron un mayor control y entendimiento del desarrollo de la solución propuesta, para clientes y desarrolladores.

Capítulo 3: Validación de la Solución

El siguiente capítulo describe el diseño de las pruebas realizadas al sistema de Notificaciones de la Plataforma de desarrollo e Integración Continua de Nova. Expone los resultados obtenidos y realiza el análisis de la escalabilidad y eficacia lograda.

3.1 Pruebas de software

Las pruebas de software consisten en la ejecución de un sistema o componente bajo condiciones o requisitos específicos, con el objetivo de observar y registrar los resultados para posteriores análisis. Las técnicas de diseño de casos de pruebas en una serie de pasos bien planificados, ofrecen una correcta construcción del software, de ahí su importancia en la obtención de productos de calidad. Alguna de las técnicas comprendidas en este proceso son las conocidas como pruebas de caja blanca y caja negra [Pressman, 2009]. La solución propuesta en la presente investigación fue sometida ambos tipos de técnicas, como parte de su proceso de validación.

3.2 Pruebas de caja blanca

Las pruebas unitarias representan la validación realizada a pequeños fragmentos de código, con el objetivo de comprobar que cada procedimiento o clase hace exactamente lo que debe hacer. De esta manera los desarrolladores pueden elaborar un código más confiable, donde se detectan fallos a medida que se programa, sin necesidad de esperar al final, lo que resulta menos engorroso y más eficiente [Pressman, 2009]. Para la implementación de la solución propuesta, del lado del servidor, las pruebas unitarias se generan mediante la herramienta Selenium IDE en su versión 2.5.0, y en la aplicación cliente mediante la biblioteca unitest de Python. De acuerdo a la metodología utilizada, todo este proceso se lleva a cabo mediante las fases de desarrollo y solo a las funcionalidades críticas. A continuación se ilustran algunos de los resultados obtenidos.

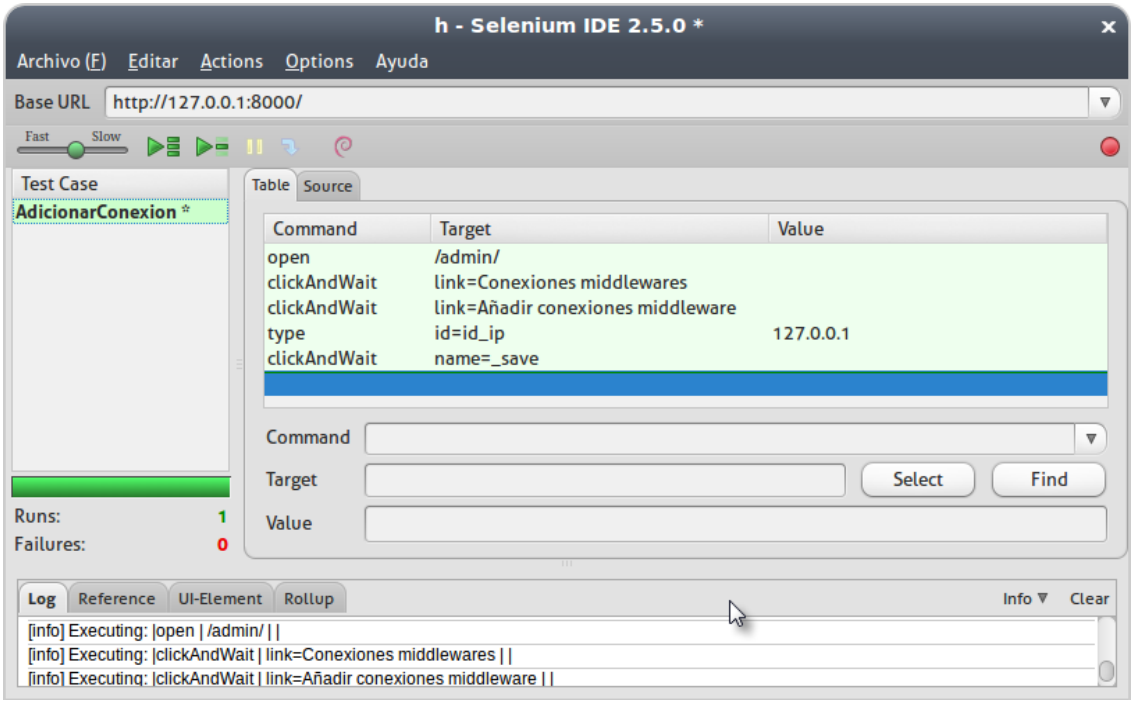

Ilustración 22: Prueba unitaria a la funcionalidad añadir conexión

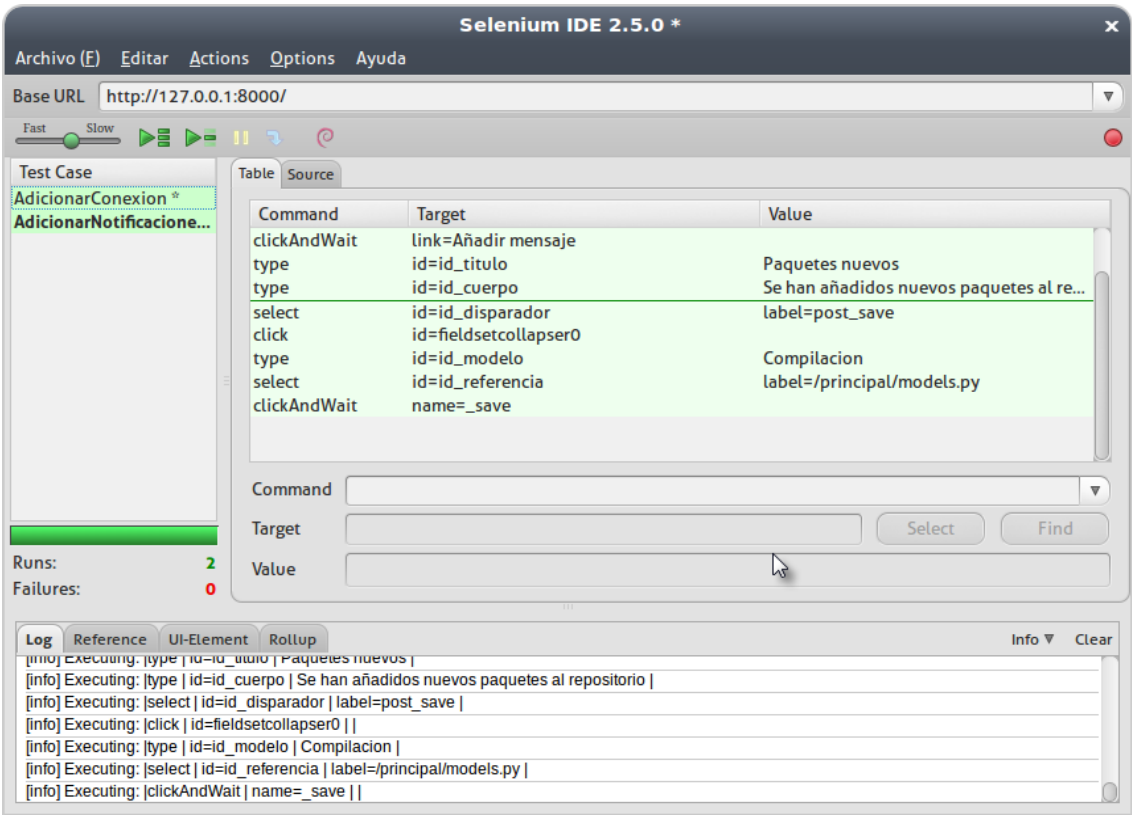

Ilustración 23: Prueba unitaria a la funcionalidad añadir notificaciones

3.3 Pruebas de caja negra

Las pruebas de caja negra representan las validaciones realizadas a las interfaces del software. Se centra en sus requisitos funcionales e intenta detectar errores mediante el estudio de las salidas obtenidas a partir de un conjunto de datos de entrada. Con el objetivo de asegurar que cada requisito de la aplicación sea revisado, debe haber al menos un caso de prueba para cada requisito. Los casos de prueba deben ser formulados antes de comenzar la prueba y poseer una entrada conocida y una salida esperada [Pressman, 2009]. A continuación se muestran los diseños de casos de pruebas realizados a los casos de usos críticos, basados en la plantilla proporcionada por el Centro de Calidad para soluciones informáticas CALISOFT.

Caso de prueba. Escenario Añadir Conexión

Escenario	Descripción	ip	estado	Respuesta del sistema	Flujo central
EC 1.1 Añadir conexión con el servidor middleware RabbitMQ con datos correctos.	El administrador introduce datos correctos para añadir una conexión.	V 127.0.0.1 V 127.0.0.1	V activado. V desactivado	La conexión es añadida a una lista de conexiones que se muestra junto a un mensaje de confirmación.	El administrador accede a la opción “Conexiones_ middleware” de la vista de administración. Se muestra el formulario para gestionar una conexión y selecciona la opción adicionar conexión. Introduce los datos y presiona el botón Grabar
EC 1.2 Añadir conexión con datos incorrectos.	El administrador introduce datos incorrectos para añadir una conexión.	I 355.67.89. I 355.67.89.	V activado V desactivado	El sistema muestra un mensaje indicando el error ocurrido.	

Caso de prueba. Escenario Editar Conexión

Escenario	Descripción	ip	estado	Respuesta del sistema	Flujo central
EC 1.3 Editar conexión con el servidor middleware RabbitMQ con datos correctos.	El administrador introduce datos correctos para modificar una conexión.	V 127.0.0.1 V 127.0.0.1	V activado. V desactivado	La conexión es añadida a una lista de conexiones que se muestra junto a un mensaje de confirmación.	El administrador accede a la opción “Conexiones_ middleware” de la vista de

EC 1.4 Editar conexión con el servidor middleware RabbitMQ con datos incorrectos.	El administrador introduce datos incorrectos para modificar una conexión.	I	V	El sistema muestra un mensaje indicando el error ocurrido.	administración. Se muestra el formulario para gestionar una conexión. Selecciona la que desea editar. Se muestra el formulario para añadir conexiones, Introduce los datos y presiona el botón Grabar.
		355.67.89.	activado		
		I	V		
		355.67.89.	desactivado		

Caso de prueba. Escenario Eliminar Conexión

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 1.5 Eliminar conexión correctamente.	El sistema muestra una lista de conexiones y elimina las conexiones seleccionadas. Esta funcionalidad no lleva variables solo se debe seguir el flujo central.	El sistema muestra un mensaje de confirmación: ¿Seguro que desea eliminar la conexión middleware 127.0.0.1?. Si el usuario selecciona el botón “Si estoy seguro” el sistema elimina la conexión y muestra un mensaje de confirmación.	El administrador accede a la opción “Conexiones_ middleware” de la vista de administración. Se muestra el formulario para gestionar una conexión. Selecciona la que desea eliminar. Selecciona la acción “Eliminar conexiones middleware”, y presiona el botón ir.

Caso de prueba. Variables Escenario Gestionar Conexión

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
Variable 1	ip	Campo de texto	NO	Dirección ip del servidor middleware RabbitMQ
Variable 2	estado	Campo de texto	NO	Estado del servidor middleware
Variable 3	puerto	Campo de texto	NO	Puerto para la conexión con el middleware RabbitMQ su valor por defecto es 5671

Caso de prueba. Escenario Añadir Notificaciones

Escenario	Descripción	titulo	cuerpo	disparador	modelo	campo	valor	Respuesta del sistema	Flujo central
EC 2.1 Añadir notificaciones con datos correctos	El administrador introduce datos correctos para añadir una notificación	V	V	V	V	V	V	El sistema añade la notificación y muestra un mensaje de confirmación.	El administrador accede a la opción “Mensajes” de la vista de administración. Se muestra el formulario para gestionar una notificación y selecciona la opción adicionar notificación. Introduce los datos y presiona el botón Grabar
		Repositorios	Se ha actualizado el repositorio de paquetes.	Post_ *save*	Repositorio => repo/model.py	Repositorio =>nombre	Nova		
		V	V	V	V	V	V		
		Repositorios	Se ha actualizado el repositorio de paquetes.	*spam*	vacío	vacío	vacío		
		Repositorios	Se ha actualizado el repositorio de paquetes.	Post_ save	Repositorio=> repo/model.py	vacío	vacío		
EC 2.2 Añadir notificaciones con datos incorrectos	El administrador introduce datos incorrectos para añadir una notificación.	I	I	I	N/A	N/A	N/A	El sistema muestra un mensaje indicando el error.	
		vacío	vacío	vacío					

Caso de prueba. Escenario Editar Notificaciones

Escenario	Descripción	titulo	cuerpo	disparador	modelo	campo	valor	Respuesta del sistema	Flujo central
EC 2.3 Editar notificaciones con datos correctos	El administrador introduce datos correctos	V	V	V	V	V	V	El sistema añade la notificación y muestra un mensaje de confirmación	El administrador accede a la opción “Mensajes” de la vista de administración. Se
		Repositorios	Se ha actualizado el repositorio de	Post_ save	Repositorio => repo/ mdel.py	Repositorio =>Nombre	Nova		

	para modificar una notifica-ción		paque-tes.						muestra el formulario para gestionar una notificación y selecciona la notificación que desea editar. El sistema muestra el formulario para adicionar una notificación. Introduce los datos y presiona el botón Grabar
		V	V	V	V	V	V		
		Reposi-torios	Se ha actuali-zado el reposi-torio de paque-tes.	*spam*	vacío	vacío	vacío		
		Reposi-torios	Se ha actuali-zado el reposi-torio de paque-tes.	Post_ save	Reposi-torio => repo/ mdel.py	vacío	vacío		
EC 2.4 Editar notifica-ciones con datos incorrec-tos	El administra dor introduce datos incorrec-tos para modificar una notifica-ción	I	I	I	N/A	N/A	N/A	El sistema muestra un mensaje indican-do el error.	
		vacío	vacío	vacío					

Caso de prueba. Escenario Eliminar Notificaciones

Escenario	Descripción	Respuesta del sistema	Flujo central
EC 2.5 Eliminar notificación correctamente.	El sistema muestra una lista de notificaciones y elimina las notificaciones seleccionadas. Esta funcionalidad no lleva variables solo se debe seguir el flujo central.	El sistema muestra un mensaje de confirmación: ¿Seguro que desea eliminar la notificación “Repositorios”?. Si el usuario selecciona el botón “Sí estoy seguro” el sistema elimina la notificación y muestra un mensaje de confirmación.	El administrador accede a la opción “Mensajes” de la vista de administración. Se muestra el formulario para gestionar una conexión. Selecciona la que desea eliminar. Selecciona la acción “Eliminar Mensajes”, y presiona el botón ir.

Caso de prueba. Variables Escenario Gestionar Notificaciones

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
Variable 1	título	Campo de texto	NO	Título de la notificación.
Variable 2	cuerpo	Campo de texto	NO	Contenido de la notificación.

Variable 3	disparador	Campo de selección	NO	Señal que activará el envío del mensaje.
Variable 4	modelo	Campo de selección	Sí	Nombre del modelo a que hace referencia el disparador y el mensaje.
Variable 5	campo	Campo de selección	Sí	Nombre de cualquier atributo de un modelo determinado.
Variable 6	valor	Campo de texto	Sí	Valor que puede tomar un campo de un modelo en un momento determinado.

Caso de prueba. Escenario Autenticar Usuarios

Escenario	Descripción	URL	Usuario	Contraseña	Respuesta del sistema	Flujo central
EC 3.1 Autenticar usuarios con datos correctos.	El usuario introduce los datos correctos para su autenticación.	V https://10.53.4.76:8000	V *admin*	V *admin*	El sistema muestra la ventana del suscriptor del cliente y le proporciona acceso a todas sus funcionalidades, guarda además los datos introducidos en el anillo de llaves de gnome.	El usuario ejecuta la aplicación cliente. Introduce el url de la plataforma de desarrollo de Nova y su usuario y contraseña.
EC 3.2 Autenticar usuarios con datos incorrectos	El usuario introduce sus datos incorrectos para su autenticación.	I ftp://10.53.4.766:800000	I jjjjj	I kkk	El sistema vuelve a mostrar la ventana de autenticación con un mensaje de error.	

Caso de prueba. Variables Escenario Autenticar Usuarios

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
Variable 1	url	Campo de texto	NO	Url de la plataforma de Desarrollo e Integración continua de Nova.

Variable 2	usuario	Campo de texto	NO	Usuario de la Plataforma de Desarrollo e Integración continua de Nova para la persona que desea autenticarse.
Variable 3	contraseña	Campo de texto	NO	Contraseña de la Plataforma de Desarrollo e Integración continua de Nova para la persona que desea autenticarse.

Caso de prueba. Escenario Suscribir usuarios

Escenario	Descripción	Lista de notificaciones	Respuesta del sistema	Flujo central
EC 4.1 Suscribir usuarios a las notificaciones.	El sistema muestra una lista con los títulos de las notificaciones existentes en la plataforma.	V Repositorios Compilación Actualizaciones.	El sistema muestra la lista de notificaciones con los temas a los que el usuario se ha suscrito marcados y los temas que no se ha suscrito desmarcados.	El usuario selecciona la opción suscribir del menú contextual del icono de notificación de la aplicación cliente.
EC 4.2 Suscribirse a los temas selecciona dos opción Aceptar.	El sistema muestra una lista con los títulos de las notificaciones existentes en la plataforma.	V Repositorios Compilación Actualizaciones.	El sistema marca las opciones seleccionadas y activa una barra de progreso manteniendo en frío toda la interfaz hasta que esta termine.	El usuario selecciona las notificaciones a las que se quiere suscribir y selecciona el botón Aceptar.
EC 4.3 Suscribirse a los temas selecciona dos opción Cancelar.	El sistema muestra una lista con los títulos de las notificaciones existentes en la plataforma.	V Repositorios Compilación Actualizaciones.	El sistema cierra la ventana del suscriptor.	El usuario, selecciona el botón Cancelar de la ventana del suscriptor.

Caso de prueba. Variables Escenario Suscribir usuarios

No	Nombre del campo	Clasificación	Valor Nulo	Descripción
Variable 1	Lista de notificaciones	Campo de selección	SI	Lista del título de las notificaciones emitidas por la plataforma.

3.4 Pruebas de carga y estrés del sistema

Para evaluar la escalabilidad y eficacia de la solución propuesta, se efectuaron las pruebas de carga y estrés, enfatizando en el flujo de datos enviados desde la plataforma hacia el middleware. Es aquí donde se produce una mayor distorsión del valor de estas variables, ya que en la entrega de los mensajes por

parte de RabbitMQ a los usuarios suscritos, constituyen conceptos altamente probados [RabbitMQ.org, 2012].

Las pruebas fueron realizadas en la herramienta Jmeter en su versión 2.3.4 diseñada para comportamiento de cargas de pruebas funcionales y la medición del rendimiento. El entorno en que fueron realizadas cumple con las siguientes características: PC Servidor con Microprocesador Intel® Core™ i3-2100 CPU @ 3.10GHz × 4 y 2 GB RAM.

Reporte generado por la herramienta Jmetter:

Funcionalidad	URL	Peticiones	Hilos concurrentes	Error	Rendimiento	Kb/sec
Enviar notificaciones	/mensajes/	50	10	0.00%	28,1/sec	7,7
Enviar notificaciones	/mensajes/	1000	200	0.020%	5,2/sec	4,7

3.4.1 Análisis de resultados

Considerando el número de notificaciones que en un momento dado, pudieran generar cada uno de los componentes de la Plataforma de desarrollo e Integración Continua de Nova, la solución propuesta fue sometida a una carga esperada de 10 hilos en ejecución que equivalen a 50 peticiones concurrentes, respondiendo con un rendimiento de 28,1 peticiones por segundo y 0.00% de error lo cual demuestra el buen rendimiento del sistema bajo situaciones normales. Finalmente las pruebas fueron repetidas pero esta vez para una carga que representa 4 veces la esperada, obteniéndose un rendimiento de 5,2 peticiones por segundo y solo un 0,020% de error. Quedando demostrada la alta capacidad que posee el sistema de notificaciones para atender un número de peticiones cada vez mayor con buen rendimiento y tolerancia a fallos, se concluye que los niveles de escalabilidad y eficacia a que se aspiraban han sido alcanzados.

Conclusiones parciales

- Las pruebas unitarias permitieron la obtención de un código legible y optimizado durante toda la etapa de desarrollo.
- Las pruebas de caja negra realizadas permitieron detectar y corregir fallos en las interfaces, garantizando una correcta validación de los datos que se introducen en el sistema.
- Las pruebas de carga y estrés realizada evaluaron la escalabilidad y eficacia del sistema de notificaciones de la Plataforma de Desarrollo e Integración Continua de Nova.

Conclusiones generales

- El estudio realizado a la infraestructura de las distribuciones destacadas arrojó la necesidad de desarrollar el sistema de notificaciones de la Plataforma de Desarrollo e Integración Continua de Nova para mantener informado a sus usuarios.
- La arquitectura genérica del sistema de notificaciones, permitió incorporarlo a la Plataforma de Desarrollo e Integración continua de Nova.
- La utilización del protocolo asíncrono de colas de mensajes AMQP y del middleware orientado a mensajes RabbitMQ permitió notificar a los usuarios de la Plataforma de Desarrollo e Integración Continua de Nova con escalabilidad y eficacia.

Recomendaciones

- Incorporar un mecanismo de plantillas para permitir al administrador generar notificaciones más personalizadas.
- Incorporar nuevas vías de notificación como por ejemplo: los canales RSS y el correo electrónico.
- Desarrollar un cliente de Notificaciones para telefonía móvil.
- Utilizar el sistema de notificación de la Plataforma de Desarrollo e Integración Continua para contribuir al desarrollo en comunidad de la distribución cubana de GNU/Linux Nova.

Referencias Bibliográficas

- [1] Alonso, Falcón Randy. Cubadebate. Cubadebate. [En línea] Consejo Editorial de Cubadebate, 9 de mayo de 2009. [Consultado el: 15 de Noviembre de 2013]. Disponible en: <www.cubadebate.cu>.
- [2] Apache ActiveMQ ™ . [En línea]. [Accedido: 09-feb-2014]. Disponible en: http://activemq.apache.org/.
- [3] Comunicating and getting help–FedoraProject, [En línea], [Consultado: 18 de Noviembre 2013]. Disponible en: <https //fedoraproject.org/wiki/Communicating _and getting_help/es>
- [4] Diccionario de la Lengua Española , [En línea], [Consultado el: 19 de Noviembre de 2013]. Disponible en: <http://buscon.rae.es>
- [5] Django documentation — Django 1.3.1 documentation. [En línea]. [Accedido: 12-feb-2014].Disponible en: file:///usr/share/doc/python-django-doc/html/index.html#.
- [6] Edwards, J., 2011. Soyuz/TechnicalDetails - Launchpad Development.[En línea]. [Consultado el 17 de diciembre de 2012].Disponible en:<https://dev.launchpad.net/Soyuz/TechnicalDetails>
- [6.1] Edwards, J., 2011. Soyuz/TechnicalDetails - Launchpad Development.[En línea]. [Consultado el 17 de diciembre de 2012].Disponible en:<https://dev.launchpad.net>
- [7] Features | AMQP. [En línea]. [Accedido: 08-feb-2014].Disponible en: <http://amqp.org/product/features>.
- [8] Hodek, R., 2011. Debian -- Autobuilder network. [En línea],[Consultado el: 17 de Diciembre 2013]. Disponible en : <http://www.debian.org/devel/buildd/index.en.html>.
- [9] Inicio - linuxparatodos.net, [En línea], [Consultado el: 18 Noviembre 2013]. Disponible en: <http://www.linuxparatodos.net/>
- [10] Jacobson I., Jacobson S., "Beyond methods and CASE : The software engineering process with its integral support enviroment" , Object Magazine, January 1995, p. 120-165.
- [11] Jacobson,I. ,Jacobson S. ,"Designing an integrated SEPSE", Object Magazine, September 1995, p.98-110.

- [12] Launchpad, [En línea], [Consultado el: 18 Noviembre 2013]. Disponible en: <https://launchpad.net/>
- [13] Llamaret, Heredia Maikel. SWL-X: Rompiendo el cascarón. [En línea]. Edición 0 (2013). [Consultado el: 12 de Noviembre del 2013]. Disponible en: <http://store.uci.cu/humanos/doc/revistas/CC/swlx0.pdf>.
- [14] Mahmoud, Qusay H. Oracle.Getting Started with Java Message Service (JMS). Oracle.Getting Started with Java Message Service (JMS). [En línea] noviembre de 2004. [Consultado el: 2 de mayo de 2014.]. Disponible en: <http://java.sun.com/developer/technicalArticles/Ecommerce/jms/>.
- [15] Marciniak, P. LINUX+ : Android en la era digital al alcance de tu dispositivo favorito.[En línea]. Edición 10 (2010)[Consultado el: 1 de Diciembre de 2013]
- [16] MQ Telemetry Transport (MQTT) V3.1 Protocol Specification». [En línea]. [Accedido: 08-feb-2014]. Disponible en: <http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html>.
- [17] Nova está poniendo a punto una nueva plataforma de desarrollo. [En línea].La Habana: blog de software libre, Dariem Pérez Herrera, 2 de Octubre del 2013.[Consultado el: 12 de Noviembre 2013] Disponible en: <http://humanos.uci.cu/2013/10/nova-esta-poniendo-a-punto-una-nueva-plataforma-de-desarrollo/>.
- [18] Novell, 2011. openSUSE:Build Service Tools - openSUSE. [En línea], [Consultado: el 17 de Diciembre 2013]. Disponible en: <http://en.opensuse.org/openSUSE:Build_Service_Tools>.
- [18.1] Novell, 2011. openSUSE:Build Service Tools - openSUSE. [En línea], [Consultado el 17 de Diciembre 2013]. Disponible en :<http://en.opensuse.org>.
- [19] Oramas, Goñi Angel . Nova 3.0, avances y expectativas de la distribución cubana de GNU/Linux. [En línea]. Vol. 4, No. 6 (2011). [Consultado el: 28 de Septiembre del 2013]. Disponible en: <http://publicaciones.uci.cu/index.php/ SC/article/view/695 >.
- [20] Offut,J,"Quality Attributes of Web Software Applications", en IEEE Software, marzo-abril de 2002. pp. 25-32.
- [21] OpenJMS - OpenJMS [En línea]. [Accedido: 09-feb-2014]. Disponible en: <http://openjms.sourceforge.net/>.

- [22] Pierra, Fuentes Allan. Nova distribución Cubana de GNU/LINUX, Tesis (Maestría en informática aplicada). La Habana, Cuba: Universidad de la Ciencias Informáticas, 2013. pp. 29.
- [23] Pressman, Roger S. Ingeniería del software. Un enfoque práctico. s.l: McGraw-Hill, 2009.
- [24] Pérez, Herrera, Dariem. Sistema distribuido para automatizar la construcción de repositorios de paquetes binarios de la distribución cubana de GNU/Linux Nova. Tesis (Maestría en Ciencias de la Computación). Santa Clara, Cuba: Universidad Central “Marta Abreu” de Las Villas, 2013. pp 25-32.
- [25] Python Software Foundation, 2009. Data-model--Python v3.0.1 documentation. [Consultado: 22 diciembre 2013]. Disponible en: <http://docs.python.org/3.0/reference/datamodel.html#special-method-names>
- [26] Quiñones Azcarate E., "Herramientas de Apoyo al desarrollo de Software" , Jornadas Técnicas con el estado, presidencia de APESOL 2006-2008.
- [27] RabbitMQ. RabbitMQ. [En línea] 2012. [Consultado el: 22 de enero del 2014.]. Disponible en: <http://www.rabbitmq.com/documentation.html>.
- [27.1] RabbitMQ. RabbitMQ. [En línea] 2012. [Consultado el: 22 de enero del 2014.]. Disponible en: <http://www.rabbitmq.com/blog/2012/04/25/rabbitmq-performance-measurements-part-2/>
- [28] Red Hat, 2012. Koji – FedoraProject. [En línea] ,[Consultado 17 Diciembre 2013]. Disponible en: <http://fedoraproject.org/wiki/Koji>.
- [29] Sivianes, Francisco, y otros. Servicios en red. Madrid: Paraninfo, 2010. 9788497327657.
- [30] STOMP. [En línea]. [Accedido: 12-feb-2014]. Disponible en: <http://stomp.github.io/>.
- [31] Visual Paradigm, 2012. UML CASE tool for software development. [En línea]. [Consultado: 8 octubre 2013]. Disponible en: <http://www.visual-paradigm.com/product/vpuml/>
- [32] Dan Rubel Eric Clayberg.Eclipse: Building Commercial-Quality Plug-ins, Second Edition .s.l.: Addison Wesley Professional, 2006.
- [33] Rob, Peter y Coronel, Carlos. Sistema de bases de datos. México: s.n., 2006. 061906209.

- [34] Centro de encuentro BPM. El libro del BPM. Madrid: s.n., 2011. 9788461483679.

- [35] Distributed Computing Made Simple - zeromq. [En línea]. [Accedido: 13-feb-2014]. Disponible en: <http://zeromq.org/>.
- [36] OpenUP. [En línea]. [Accedido: 13-feb-2014]. Disponible en: <http://epf.eclipse.org/wikis/openup/index.htm>.

- [37] PyDev. [En línea]. [Accedido: 12-feb-2014]. Disponible en: <http://pydev.org/>.
- [38] JSON. [En línea]. [Accedido: 02-mar-2014]. Disponible en: <http://www.json.org/>.

- [39] DistroWatch.com: Nova. [En línea]. [Accedido: 02-mar-2014]. Disponible en: <http://distrowatch.com/table.php?distribution=Nova>.

- [40] Oliver Gómez, Salvador ADT Plugin | Android Developers. [En línea]. [Accedido: 02-mar-2014]. Disponible en: <http://developer.android.com/tools/sdk/eclipse-adt.html>.

- [41] Manual de Java. [En línea] [Citado el: 3 de febrero de 2010.]. [Disponible en] <http://manual-java.com/manualjava/caracteristicas-java.html>.

- [42] PyQt4 Reference Guide — PyQt 4.10.3 Reference Guide. [En línea]. [Accedido: 03-mar-2014]. Disponible en: http://pyqt.sourceforge.net/Docs/PyQt4/.

- [43] Python. PEP 8 - Style Guide for Python Code. [En línea] [Accedido: 14 de Enero de 2013].Disponible en: < http://www.python.org/dev/peps/pep-0008/>.

- [44] Holovaty, Adrian y Kaplan Jacob. The Definitive Guide to Django: Web Development Done Right. Apress, Diciembre de 2007.
- [45] Gonzales, Duque Raúl. Python para todos. España , Mayo del 2007.

- [46] Larman, Crai. UML y Patrones: Una introducción al análisis y diseño orientado a objeto y al proceso unificado 2da edición. España, 2003

- [47] Videla, Alvaro y Williams Jason. RabbitMQ in Action. Estados unidos, 2012.

Tablas

Caso de uso: Enviar notificaciones

Objetivo	Enviar la notificación emitida por los sistemas y herramientas que componen la Plataforma de Desarrollo e Integración continua de Nova hacia el servidor middleware.	
Actores	Administrador.	
Resumen	El caso de uso se inicia si una vez que el Administrador ha añadido una notificación él o algunos de los sistemas o herramientas activan su disparador.	
Complejidad	Media.	
Prioridad	Crítico.	
Precondiciones	Debe haber notificaciones añadidas.	
Postcondiciones	Se envía la notificación al middleware.	
Flujo de eventos		
Flujo básico <Enviar Notificaciones>		
	Actor	**Sistema**
1		Captura el disparador.
2		Busca las notificaciones referentes al modelo que activa el disparador.
4		Envía la notificación al middleware.
5		Termina el caso de uso.
Relaciones	**CU Extendido**	Enviar notificaciones en el CU Gestionar notificaciones.
Requisitos no funcionales	Debe estar instalada la biblioteca pika 0.9.8 de Python.	

Caso de uso: Suscribir usuarios

Objetivo	Suscribir a los usuarios a los temas de su interés.	
Actores	Usuario.	
Resumen	El caso de uso se inicia cuando el usuario ejecuta la aplicación cliente y sus credenciales han sido validadas.	
Complejidad	Baja.	
Prioridad	Crítico.	
Precondiciones	El usuario debe estar autenticado.	
Postcondiciones	Se le muestra a los usuarios solo las notificaciones seleccionadas por ellos.	
Flujo de eventos		
Flujo básico <Suscribir usuarios>		
	Actor	**Sistema**
1	El usuario ejecuta la aplicación cliente.	Valida las credenciales del usuario y sin son correctas, muestra la ventana del suscriptor con la lista de los temas de la notificaciones.
2	El usuario selecciona los temas sobre los que desea ser notificado.	Guarda la configuración de los temas suscritos.

3		Entra en un ciclo constante a la escucha de los mensajes provenientes del *middleware.*
4		Termina el caso de uso.
Requisitos no funcionales	Debe estar instalada la biblioteca pika 0.9.8 de Python	
Prototipos	Ver Anexo 3	

Caso de uso: Mostrar Notificaciones

Objetivo	Informar a los usuarios de lo que acontece en la Plataforma de Desarrollo e Integración continua de Nova.	
Actores	Usuario.	
Resumen	El caso de uso se inicia cuando el usuario ejecuta la aplicación cliente y sus credenciales han sido validadas.	
Complejidad	Baja.	
Prioridad	Crítico.	
Precondiciones	El usuario debe estar autenticado.	
Postcondiciones	Se le muestra la información a los usuarios.	
Flujo de eventos		
Flujo básico <Mostrar Notificaciones>		
	Actor	**Sistema**
1	El usuario ejecuta la aplicación cliente.	Valida las credenciales del usuario y sin son correctas, entra en un ciclo constante a la escucha de los mensajes provenientes del *middleware*.
2		Cuando un mensaje es recibido, muestra su título y cuerpo mediante un *tooltip* al usuario teniendo de acuerdo a los temas que se ha suscrito.
3		Si el mensaje es de actualización, entonces actualiza la lista de notificaciones y la conexión del middleware.
4	El usuario cancela la ejecución de la aplicación cliente.	Termina el caso de uso.
Relaciones	**CU Extendidos**	Actualizar: Paso 3 del Flujo Básico.
Requisitos no funcionales	Debe estar instalada la biblioteca pika 0.9.8 de Python.	

Caso de uso: Actualizar

Objetivo	Actualizar la lista de notificaciones y la conexión del middleware en la aplicación cliente.
Actores	Usuario.
Resumen	El caso de uso se inicia cuando el usuario ejecuta la aplicación cliente y sus credenciales han sido validadas, o cuando el CU mostrar notificaciones recibe

	una notificación de actualización.	
Complejidad	Media.	
Prioridad	Crítico.	
Precondiciones	El usuario debe estar autenticado.	
Postcondiciones	Se actualizan la lista de notificaciones y la conexión middleware.	
Flujo de eventos		
Flujo básico <Mostrar Notificaciones>		
	Actor	**Sistema**
1	El usuario ejecuta la aplicación cliente y se autentica.	Una vez validadas las credenciales actualiza la lista de notificaciones y la conexión del middleware. El sistema entra en un ciclo constante a la escucha de los mensajes provenientes del *middleware*.
2		Cuando un mensaje de actualización es recibido nuevamente es actualizada la lista de notificaciones y la conexión con el middleware.
3	El usuario cancela la ejecución de la aplicación cliente.	Termina el caso de uso.
Requisitos no funcionales	Debe estar instalada la biblioteca pika 0.9.8 de Python.	

Anexo 1

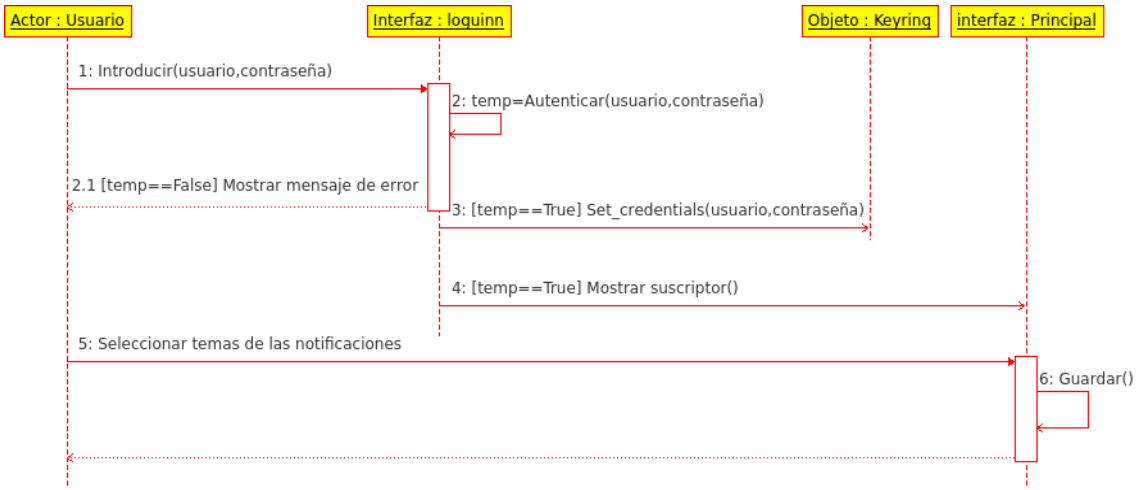

Ilustración 24: Diagrama de secuencia CU Suscribir usuarios.

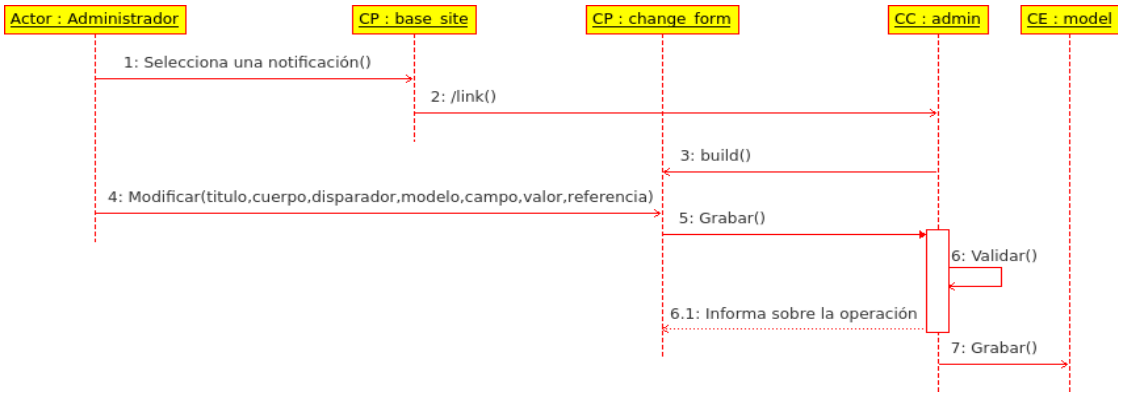

Ilustración 25: Diagrama de secuencia CU Mostrar notificaciones.

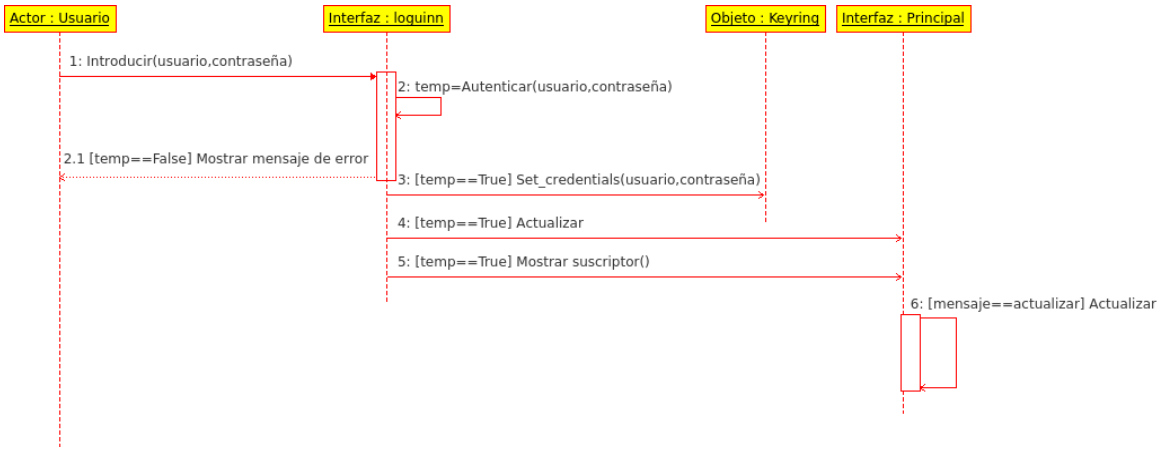

Ilustración 26: Diagrama de secuencia CU Actualizar.

Anexo 2

Ilustración 27: Prototipo de interfaz de usuario CU Gestionar Conexión

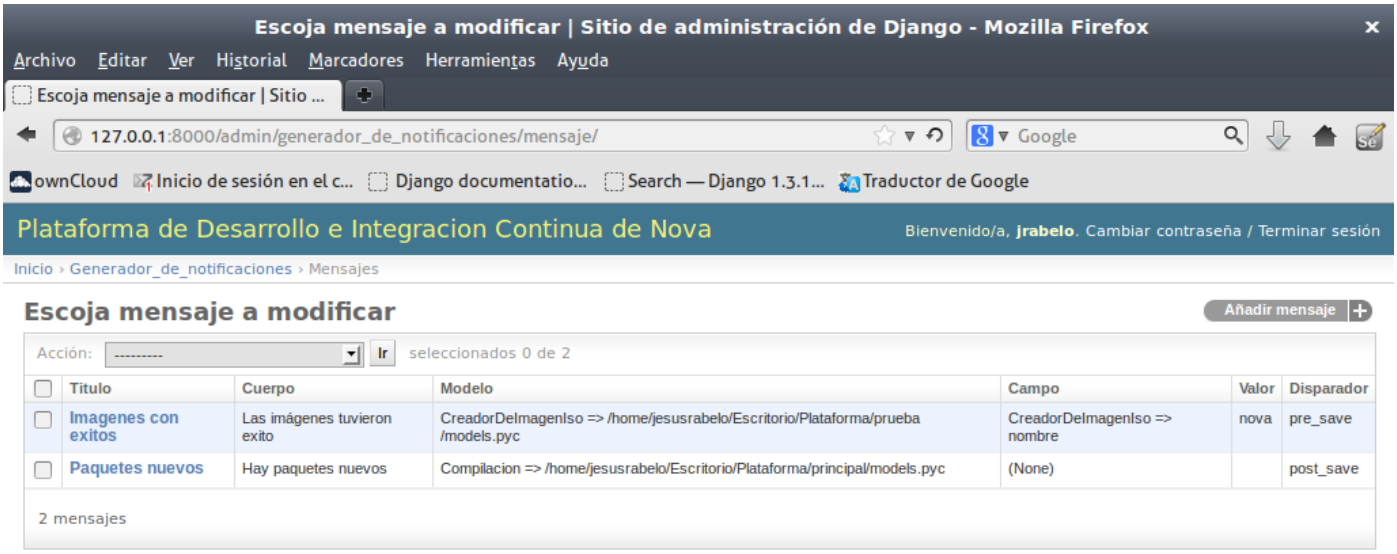

Ilustración 28: Prototipo de interfaz de usuario CU Gestionar Notificaciones

Anexo 3

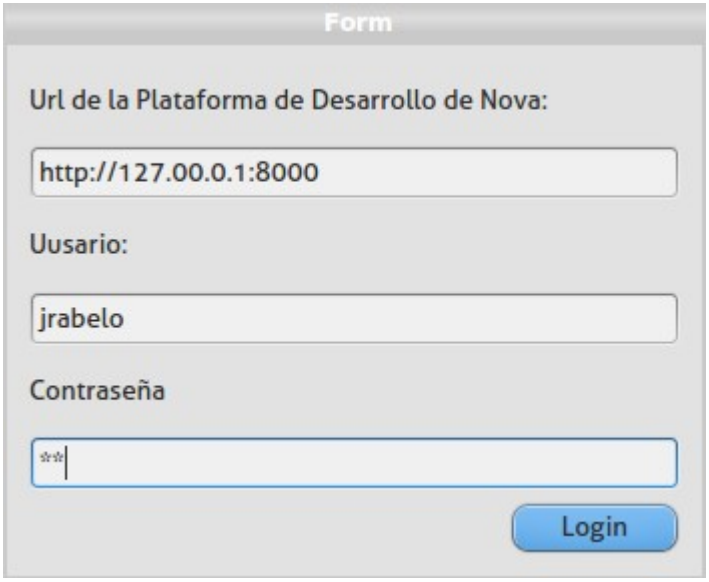

Ilustración 29: Prototipo de interfaz de usuario CU Autenticar Usuarios

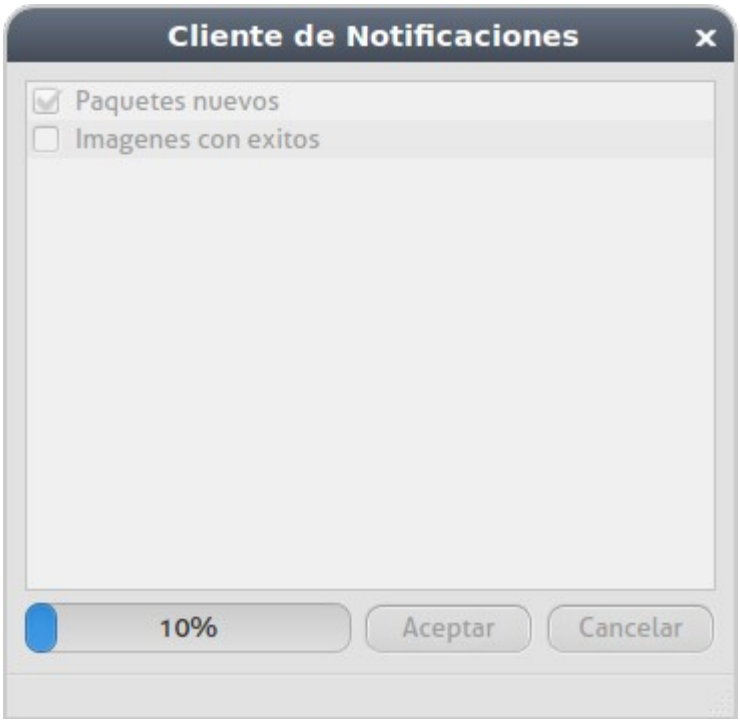

Ilustración 30: Prototipo de interfaz de usuario CU Suscribir Usuarios