

# Trabajo Profesional: Parallel-Editor

Ciancio Alessio, Mauro Lucas  
Gilioli, Leandro Ezequiel  
{maurociancio,legilioli}@gmail.com

2do cuatrimestre de 2010

## Resumen

En el presente trabajo se pretende encontrar y mostrar una solución al problema de la edición concurrente de documentos en tiempo real y distribuida. Este escenario se plantea en diversas áreas incluyendo el desarrollo de software. La técnica *pair-programming*, generalmente propuesta por las metodologías ágiles, puede verse beneficiada al usar el producto que se implementa en este trabajo profesional.

Se desarrolló una solución que permite la edición simultánea de uno o más documentos, asegurando que no exista divergencia entre las partes aún si la conexión entre ellos posee alta latencia. A su vez, se utiliza un mecanismo optimista para modificar los documentos, en el cual no se requiere de un árbitro central que coordine las partes involucradas.

Por último, la solución se integra con el IDE Eclipse mediante un plugin desarrollado para este trabajo profesional facilitando la aplicación de la técnica antes mencionada.

# Visión Trabajo Profesional

Ciancio Alessio, Mauro Lucas  
Gilioli, Leandro Ezequiel

2do cuatrimestre de 2009

## 1. Objetivo

El objetivo de este documento es analizar y definir los requerimientos y necesidades de alto nivel para el proyecto a presentarse como Trabajo Profesional.

## 2. Alcance

El proyecto consiste en un sistema de edición de documentos de texto plano multiusuario que permita a varios usuarios confeccionar un documento en forma colaborativa.

Los usuarios editarán en tiempo real un mismo documento que es compartido por uno de ellos o que está en un servidor centralizado. Los cambios que introduce un usuario serán visibles instantaneamente por los demás.

El nombre del proyecto es *Parallel Editor*.

Los usuarios de este proyecto son aquellas personas que trabajen produciendo documentos de código fuente o texto plano en general, como por ej. desarrolladores de software, diseñadores de páginas web, estudiantes realizando trabajos prácticos, etc.

Cada día es más común la necesidad de trabajar en conjunto, estar en contacto con colegas de forma remota y desarrollar en equipo en forma distribuida. Este proyecto les puede aportar un ahorro de tiempo considerable al poder ver, editar y mejorar un documento por varios usuarios a la vez.

Hoy en día los entornos de desarrollo impuestos como estándar de facto proveen escaso soporte para este tipo de soluciones. Se pretende lograr una solución que no obligue al usuario a cambiar las herramientas que actualmente maneja, sino proveer una integración con las mismas a fin de disminuir la curva de aprendizaje.

### 3. Posicionamiento

El problema surge de la necesidad de comunicación y retroalimentación constante. Los usuarios necesitan que su trabajo sea revisado por sus colegas de forma de asegurar calidad y estándares. Para este tipo de trabajo es indispensable que la herramienta sea eficiente y los documentos se compartan instantáneamente y puedan mejorarse colaborativamente.

#### 3.1. Usuarios

Como se nombró anteriormente, existen diversos tipos de usuarios en este proyecto:

- Los principales son los desarrolladores de cualquier lenguaje. Les permitirá compartir con sus pares las soluciones a problemas, requerir ayuda en determinados temas, aumentar la productividad obteniendo un *feedback* mas rápidamente, etc.
- Desarrolladores de páginas web: debido a que la mayoría de los formatos de las páginas web son formatos de texto.
- Desarrollo de documentos de texto: documentos basados en texto o en Latex.
- Estudiantes: para el desarrollo de trabajos prácticos.

#### 3.2. Interesados

A continuación se revisará la lista de interesados:

- Compañías de IT cuya principal actividad es el desarrollo de software.
- Equipos de trabajo con alto nivel de concurrencia sobre un mismo documento.
- Equipos de trabajo geográficamente distribuidos.
- Entornos de desarrollo integrados (IDE) con arquitectura de soporte de plugins.

El problema impacta fuertemente en ambientes o situaciones en las cuales se requiere la elaboración de un documento de forma colaborativa y en tiempo real.

Una solución a este problema es la que se pretende mostrar en este documento.

## 4. Descripción general del Producto

La solución permitirá a usuarios que estén conectado entre sí a través de una LAN o Internet compartir un documento de texto e ir aportando contenido al mismo en tiempo real. En el escenario más típico un usuario denominado *host* o anfitrión creará un servicio en la red para que los restantes puedan ingresar y comenzar el desarrollo.

Una vez establecida la sesión de desarrollo todos los usuarios pueden aportar información al documento en tiempo real y al mismo tiempo observar los cambios introducidos por los demás. La solución tendrá soporte para evitar problemas de concurrencia y para mantener la consistencia del documento.

En cualquier momento usuarios pueden ingresar o dejar la sesión de desarrollo. Los nuevos usuarios obtendrán la versión actual del documento que se encuentran editando.

Varios usuarios podrán colaborar en varios documentos simultáneamente, no estando limitado sólo a trabajar en un documento a la vez.

### 4.1. Módulos

Se realizarán tres módulos principales:

#### 4.1.1. Núcleo

El primer módulo será el núcleo o *kernel* de la solución. Este módulo será el encargado de comunicarse con los integrantes de la sesión, enviar los cambios ingresados, crear y borrar documentos, crear y borrar sesiones de desarrollo, etc.

#### 4.1.2. Interfaz Gráfica

El segundo módulo será una interfaz gráfica que le permitirá al usuario realizar las operaciones desde un alto nivel. Opciones para crear sesiones de desarrollo, invitar usuarios a la sesión, sacar usuarios de la sesión, posibilidad de compartir mensajes privados entre los usuarios, abrir y cerrar documentos, ver las revisiones del documento que se está editando, etc.

#### 4.1.3. Integración con IDE

El tercer módulo realizará la integración con un entorno de desarrollo preexistente. Existen excelentes herramientas para realizar diversos proyectos en Java, C, C++, Python, etc; ya que no se busca reinventar la rueda, este módulo se integrará con esas herramientas de forma tal de aprovechar sus beneficios. Por integración se entiende editar un archivo concurrentemente en el mismo IDE aprovechando todas sus funcionalidades (generación

automática de código fuente, *refactor*, etc.).

Cabe aclarar que los dos últimos módulos son independientes entre sí y pueden funcionar por separado.

#### **4.1.4. Integración con SCM**

Además se realizará un módulo más pequeño que permita integrar la solución con sistemas de control de versiones (SCM: *source code management*) (tales como SVN, GIT o Mercurial). Este módulo permitirá:

- Luego de una sesión de desarrollo versionar el o los archivos modificados en el SCM incluyendo en el *log* quienes fueron los que trabajaron.

Como funcionalidades extras existen las siguientes:

- Poder revisar el historial de cambios del documento que se compartió y oportunamente revertir algún cambio en el mismo.
- Poder compartir mensajes instantáneos entre los usuarios.
- Interfaz web para poder ver el documento que se está editando, que usuarios están editando y quienes lo hicieron anteriormente.

# Propuesta de Trabajo Profesional

Ciancio Alessio, Mauro Lucas  
Gilioli, Leandro Ezequiel  
{maurociancio,legilioli}@gmail.com

2do cuatrimestre de 2010

## Índice

<b>1. Equipo de trabajo</b>	<b>3</b>
<b>2. Contexto</b>	<b>3</b>
<b>3. Objetivo del proyecto</b>	<b>3</b>
<b>4. Necesidades del cliente / Mercado</b>	<b>4</b>
<b>5. Competencia</b>	<b>4</b>
5.1. Google Docs [4] y Google Wave [5] . . . . .	4
5.2. BeWeeVee [6] . . . . .	5
5.3. COLA [7] - Eclipse Plugin . . . . .	5
5.4. Comparación con la presente propuesta . . . . .	5
<b>6. Alcance</b>	<b>5</b>
6.1. Núcleo . . . . .	5
6.2. API Cliente . . . . .	6
6.3. Interfaz gráfica . . . . .	6
6.4. Integración con IDE . . . . .	6
<b>7. Requerimientos funcionales</b>	<b>6</b>
7.1. Núcleo . . . . .	6
7.2. Cliente . . . . .	7
7.3. GUI . . . . .	7
7.4. Plugin . . . . .	8

<b>8. Requerimientos no funcionales</b>	<b>8</b>
8.1. Integrabilidad . . . . .	8
8.2. Seguridad . . . . .	8
8.3. Confiabilidad . . . . .	8
8.4. Portabilidad . . . . .	9
<b>9. Estimación</b>	<b>9</b>
<b>10. Planificación</b>	<b>10</b>
<b>11. Entregables</b>	<b>11</b>
<b>12. Metodología de desarrollo</b>	<b>11</b>
12.1. Roles . . . . .	12
12.2. Reuniones . . . . .	12
12.3. Documentación . . . . .	13
<b>13. Tecnologías</b>	<b>13</b>
<b>14. Infraestructura necesaria</b>	<b>14</b>
<b>15. Criterios de calidad</b>	<b>15</b>
<b>16. Riesgos</b>	<b>16</b>
<b>17. Glosario</b>	<b>17</b>

## 1. Equipo de trabajo

A continuación se listan los alumnos que desarrollaron esta propuesta y se incluye sus datos personales:

- Ciano Alessio, Mauro Lucas.  
Padrón 86.357.
- Gilioli, Leandro Ezequiel.  
Padrón 86.075.

El profesor tutor de este trabajo profesional es el Lic. Pablo Cosso.

## 2. Contexto

En el presente documento se definirá el plan de proyecto para la propuesta realizada en el documento de visión titulado “Visión Trabajo Profesional” (para más información ver [1]).

Dentro del mismo se definirán entre otros los siguientes puntos: alcance, planificación, riesgos y criterios de calidad.

Este documento servirá de soporte en el proceso de desarrollo del producto y definirá criterios para medir el avance del proyecto de forma objetiva.

## 3. Objetivo del proyecto

El objetivo del proyecto es obtener un producto de software que permita a los usuarios elaborar documentos de texto de forma concurrente y en tiempo real. El producto se dividirá en dos subproductos de menor tamaño: el primero permitirá la edición colaborativa de texto usando un software independiente de cualquier otra aplicación. El segundo comprenderá la integración con un entorno de desarrollo integrado.

El nombre producto será “Parallel Editor”.

A su vez, se pretende desarrollar el producto siguiendo los criterios de calidad que se definen más adelante en este documento.

También se desarrollará documentación de usuario y técnica del producto de modo que sea posible la extensión e incorporación a otras plataformas e IDEs.

El producto será desarrollado teniendo en cuenta la integración del mismo en otros contextos, por lo que se podrá observar una vez finalizado que los dos subproductos reutilizan un gran porcentaje del código fuente desarrollado una



única vez. Este punto es importante y es considerado en los requerimientos no funcionales.

## 4. Necesidades del cliente / Mercado

Se necesita una herramienta con las siguientes características:

- La posibilidad de que dos o más personas editen un mismo documento en tiempo real sin la necesidad de estar físicamente en el mismo lugar.
- Provisión al usuario de un mecanismo sencillo para la iniciación de una sesión de edición.
- Bajo costo de infraestructura al no requerir un servidor central dedicado para este producto.
- Resolución transparente de los posibles conflictos de la naturaleza concurrente de la edición en tiempo real.
- Independencia de una conexión a Internet para su disponibilidad. Esto se traduce a que el producto pueda funcionar en una red LAN que no esté conectada a internet.

## 5. Competencia

En esta sección se describirán productos existentes en el mercado y se compararán con la presente propuesta:

### 5.1. Google Docs [4] y Google Wave [5]

Provee la posibilidad de editar en tiempo real concurrentemente documentos con formato utilizando un navegador compatible. Posibilita la participación de múltiples usuarios en línea.

Sin embargo, su mayor desventaja consiste en la dependencia de una conexión a internet para la disponibilidad del servicio. No es posible hasta ahora, la utilización del mismo en una red LAN privada.

Además es sólo utilizable a través de la interfaz web del navegador por lo cual su integración con herramientas de terceros no es posible. Cabe aclarar que Google ha abandonado el desarrollo de Google Wave.

## 5.2. BeWeeVee [6]

Framework para la integración de funcionalidades de colaboración en tiempo real en aplicaciones. Se provee como un software development kit para el desarrollo sobre la plataforma .NET. Por este motivo, si bien abarca gran cantidad de desarrollos que hacen uso de dicha plataforma, no cubre totalmente el espacio de potenciales aplicaciones ya que las aplicaciones que no se desarrollan en .NET no pueden integrarlo.

Es de licencia libre para uso académico y aplicaciones open source, aunque tiene un costo para la integración en desarrollos privados.

## 5.3. COLA [7] - Eclipse Plugin

COLA es un plugin para la integración con Eclipse que permite la colaboración en tiempo real de los usuarios para editar un mismo documento de código fuente. Está desarrollado en Java y está basado en el proyecto Eclipse Communication Framework (ECF). Su principal desventaja es la dependencia con el mismo y por otra parte, limita la cantidad de usuarios que pueden participar en una sesión de edición de código a dos participantes.

## 5.4. Comparación con la presente propuesta

El producto a desarrollar estará orientado a cubrir aquellos aspectos que las soluciones antes descritas dejan de lado. Se pretenden liberar el código fuente bajo una licencia open-source, garantizar la portabilidad del mismo usando tecnologías que corren sobre la máquina virtual de Java, lograr independencia sobre otros frameworks o componentes de software y lograr que pueda haber mas de dos participantes en la misma sesión de edición.

Estas características descritas determinarán los requerimientos funcionales y no funcionales que se describirán mas adelante.

# 6. Alcance

En la visión del proyecto se listaron las posibles funcionalidades que podría incluir el producto, de ellas se seleccionarán las siguientes para ser completadas en el presente proyecto:

## 6.1. Núcleo

Consiste en una biblioteca de software que proveerá servicios de apertura y cierre de sesiones de desarrollo. Se encargará de la administración de los

participantes de la sesión de desarrollo, recibiendo e informando los cambios introducidos por cada uno de ellos y resolviendo los conflictos que se produzcan debido a la concurrencia.

También el núcleo ofrecerá una API de modo que las aplicaciones que requieran un servidor embebido puedan usar la misma.

## **6.2. API Cliente**

Comprenderá la biblioteca que se encarga de implementar el protocolo de comunicación de un participante de la sesión con el núcleo del sistema. Esta API será utilizada tanto por el cliente GUI como también por el plugin.

## **6.3. Interfaz gráfica**

Se desarrollará una pequeña aplicación independiente que permitirá la edición colaborativa utilizando las APIs antes mencionadas.

## **6.4. Integración con IDE**

Se desarrollará un agregado o plugin para integrar la funcionalidad de desarrollo colaborativo en tiempo real dentro del IDE Eclipse.

# **7. Requerimientos funcionales**

Se separarán los requerimientos funcionales según el módulo al que correspondan.

## **7.1. Núcleo**

1. Resolución de conflictos: implementación del algoritmo de resolución de conflictos. Se utilizará el algoritmo Júpiter [2] para resolver el escenario concurrente presente en el producto.
2. Documentos: implementación de las entidades que mantienen el estado del documento en memoria, mientras los participantes aplican operaciones sobre ellos para agregar o borrar contenido. El núcleo ofrecerá servicios para crear documentos vacíos o para asignar un contenido inicial al documento. El origen de este documento dependerá de la aplicación cliente.

3. Participantes: los participantes son los usuarios conectados y los cuales pueden observar los documentos que existen. No existirá restricción en cuanto a qué usuarios pueden editar cuales documentos, sino que cualquier usuario podrá suscribirse a un documento para editarlo o ver su contenido.

## **7.2. Cliente**

1. Implementación API Cliente: diseñar e implementar la API que se le ofrecerá a las aplicaciones cliente para que hagan uso de los servicios ofrecidos por el producto. A su vez, es necesario implementar el protocolo de comunicación con el núcleo para el envío y recepción de mensajes en ambos sentidos.
2. Notificación de eventos: ofrecer una abstracción a la aplicación que use esta API para manejar los eventos hacia y desde el núcleo. Los eventos que se enviarán son los correspondientes al ingreso de texto en la aplicación cliente y los eventos que se reciben son los que se originan en los otros participantes de la sesión de edición.
3. A su vez se deberán atender otro tipo de eventos tales como: mensajes de chat, mensajes de ingreso o egreso de algún participante a la sesión de edición, etc.

## **7.3. GUI**

Se requiere una aplicación gráfica que permita las siguientes funcionalidades:

1. Crear sesiones de edición.
2. Conectar a una sesión de edición.
3. Ver documentos disponibles para edición una vez que se ha conectado a un servidor.
4. Crear documento de texto vacío.
5. Compartir documento local para edición colaborativa con otros participantes.
6. Guardar el documento en edición.
7. Editar un documento de texto.

8. Listar usuarios que se encuentran en la sesión de edición.
9. Desconectar de la sesión de edición.

## **7.4. Plugin**

Se requiere un módulo plugin para la incorporación a un entorno integrado de desarrollo de las siguientes funcionalidades:

1. Compartir documento actual para edición colaborativa.
2. Conectarse a sesión de edición.
3. Desconectarse de sesión de edición.
4. Editar el documento presente en el IDE.

# **8. Requerimientos no funcionales**

## **8.1. Integrabilidad**

Se garantizará implementando dos interfaces de usuario distintas (GUI y Plugin) que hagan uso del mismo código base, solamente desarrollando las funcionalidades necesarias o distintas para cada caso. Por otra parte, se definirá y documentará una API de modo que pueda ser incorporado a otros productos por terceras partes.

## **8.2. Seguridad**

El producto para esta primer versión utilizará un protocolo de texto plano, en el cual toda la información transmitida entre los clientes no será cifrada. Si se pretende transmitir información sensible se deberá añadir una capa que permita el cifrado de la comunicación. Se buscará que esta capa adicional de seguridad sea fácilmente incorporable al producto.

## **8.3. Confiabilidad**

Los usuarios deberán poseer una versión consistente del documento que se encuentran editando. Al finalizar la sesión de edición todos los participantes deben poseer la misma versión del documento. Para garantizar este requerimiento se realizarán pruebas unitarias exhaustivas sobre el núcleo de la aplicación, ya que éste es el encargado de aplicar el mecanismo de resolución de conflictos.

## 8.4. Portabilidad

Se buscará hacer que el desarrollo sea compatible con las principales plataformas, para esto se realizará sobre la máquina virtual de Java aprovechando que cuenta con implementaciones para distintas arquitecturas.

## 9. Estimación

Se realiza una estimación de esfuerzo para el producto con el objetivo de fijar una fecha de liberación. La herramienta que se utilizó fue WBS.

En cada tarea detallada en la WBS se tuvo en cuenta el esfuerzo necesario de análisis, desarrollo, documentación y testing.

En la figura 1 se puede observar la WBS para el producto. Se observan las columnas que muestran el nombre de la tarea y la estimación de esfuerzo para la misma.

	Task Name	Work
1	<input type="checkbox"/> <b>Kernel</b>	<b>110 hrs</b>
2	API	15 hrs
3	Concurrencia	40 hrs
4	Documentos	25 hrs
5	Participantes	30 hrs
6	<input type="checkbox"/> <b>Cliente</b>	<b>45 hrs</b>
7	API	25 hrs
8	Protocolo	20 hrs
9	<input type="checkbox"/> <b>Interfaz Grafica</b>	<b>100 hrs</b>
10	Arquitectura base	15 hrs
11	Ventana de conexion	10 hrs
12	Listado de documentos	10 hrs
13	Eventos teclado	15 hrs
14	Evento actualizar vista documento	25 hrs
15	Listado usuarios	10 hrs
16	Menus aplicacion	15 hrs
17	<input type="checkbox"/> <b>Plugin</b>	<b>135 hrs</b>
18	Investigacion	50 hrs
19	Compartir documento	20 hrs
20	Acceder a sesion remota	20 hrs
21	Ventana de edicion	30 hrs
22	Listado documentos compartidos	15 hrs

Figura 1: Estimación de esfuerzo

El valor final de la estimación es 390 horas hombres (HH). Las horas hombre son horas hombres productivas.

## 10. Planificación

Se cuenta con dos recursos con una disponibilidad de 4HH diarias resultando en un total de 20HH/recurso/semana.

La estimación es de 390HH productivas, y se supone que cada 4HH de desarrollo 3HH son productivas. Por lo tanto la relación que surge es de 4/3 y al valor estimado se le agrega un 33 % más en concepto de horas reales y horas productivas.

Por lo tanto, el valor que resulta es:  $390\text{HH} + 33\% \text{ de } 390\text{HH} = 520 \text{ HH}$ .

Por último, a este valor se le agrega un 20 % en concepto de estabilización y cierre de bugs pendientes al finalizar los sprints planificados.

$520\text{HH} + 20\% \text{ de } 520\text{HH} = 625\text{HH}$ .

La estimación final de desarrollo es de 625HH.

A continuación se muestra un diagrama de Gantt (figura 2).

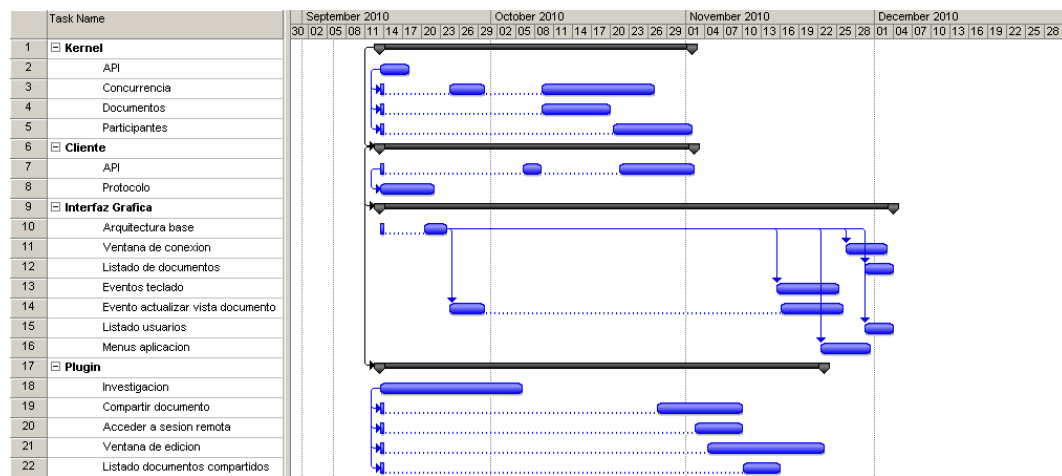


Figura 2: Gantt

El gráfico no incluye el buffer agregado al final. Se planificaron 3HH/dia/recurso productivas y el diagrama fue armado siguiendo este valor. La jornada de cada día fue realizada con esa cantidad de horas.

Este diagrama representa una guía para la organización y planificación de las tareas que serán llevadas a cabo para completar el proyecto. La planificación definitiva de las mismas se realizará con mayor precisión al inicio

de cada sprint (sprint planning meeting) incluyendo primero aquellas tareas que agreguen mayor valor en cada iteración y generen una versión usable del producto al finalizar cada una de las mismas.

Los sprints tendrán una duración de tres semanas. Observando el tiempo total estimado de la duración del proyecto se concluye que se requerirán cuatro sprints para completar el mismo.

## 11. Entregables

- Manual de usuario: contendrá la descripción y la forma de utilización de las funcionalidades de la GUI y el Plugin. Para los otros módulos se entregará la documentación técnica referente a la API.
- Manual técnico: se desarrollará el manual técnico para los módulos Núcleo y Cliente. Se incluirá documentación sobre la arquitectura, despliegue, los principales módulos, principales puntos de extensión.
- Plan de proyecto: el presente documento y la información de avance para cada sprint.
- Software: el producto listo para despliegue en CD.

## 12. Metodología de desarrollo

Para el desarrollo del proyecto se aplicará la metodología Scrum [3].

Scrum es una metodología ágil de desarrollo de software. Se enfoca principalmente en entregar la mayor cantidad de valor al negocio en el tiempo más corto posible teniendo en cuenta las prioridades del cliente. Permite obtener versiones funcionales y operativas del producto en intervalos cortos de tiempo posibilitando el retorno de la inversión del cliente con cada iteración.

En cada iteración se realizan tareas de diseño, desarrollo y prueba. El equipo de trabajo consta de un conjunto de profesionales auto-organizados que determinan cual es la mejor forma de completar las tareas para entregar las funcionalidades mas prioritarias.

Cada 2 a 4 semanas se puede ver un incremento funcional en el producto que está listo para usar y puede decidirse liberarlo en ese momento o continuar mejorándolo en otra iteración. La longitud del sprint se fija en un período tal que sea posible dejar posibles cambios afuera.



## 12.1. Roles

Dentro del equipo de trabajo típico de Scrum, existen los siguientes roles:

- *Product Owner*: decide cuales son las características del producto. Es el responsable de definir cuales serán las prioridades de la iteración tratando de maximizar el retorno de la inversión. Aprueba o rechaza el trabajo realizado.
- *Equipo Scrum*: es el equipo auto-organizado multidisciplinario de personas que se encargan de la ejecución de las tareas que se planifican para cada sprint.
- *Scrum Master*: es el responsable de la administración del proyecto. Se asegura que se apliquen los principios de Scrum durante el desarrollo de mismo. Se asegura que el equipo de trabajo sea totalmente funcional y productivo. Aísla al equipo de interferencias externas y elimina los impedimentos para que puedan completar sus tareas.

## 12.2. Reuniones

Durante el desarrollo del proyecto se realizan tres tipos de reuniones:

- *Sprint planning*: se realiza al inicio de cada sprint. Participan el product owner, scrum master y el equipo de trabajo. Se realiza la priorización y selección de las tareas y funcionalidades que serán desarrolladas durante el sprint a iniciar. Al final de esta reunión se obtiene el objetivo y la planificación del sprint.
- *Daily scrum meeting*: la realizan el equipo de trabajo junto con el scrum master. Es una reunión corta y lo mas frecuente posible (idealmente diaria). Tiene como objetivo poner al tanto al equipo de trabajo de las tareas que está realizando cada integrante y de los problemas que surgieron.
- *Sprint Review*: Se realiza al final de cada sprint. El equipo muestra el resultado del trabajo desarrollado. Toma generalmente la forma de una demostración de las funcionalidades logradas. Participan el cliente, product owner, scrum manager y el equipo scrum.
- *Sprint Retrospective*: participan exclusivamente los miembros del equipo Scrum. Se exponen lecciones aprendidas a lo largo del sprint que terminó y se toman decisiones sobre aspectos a conservar o cambiar de la forma de trabajo.

### 12.3. Documentación

Al finalizar cada sprint se dispondrá de la siguiente documentación.

- Informe de avance.
- Funcionalidades desarrolladas.
- Indicador de cobertura de la prueba. Detalle de que porcentaje y que partes del código fuente fueron alcanzados por los test unitarios.
- Estado de los tests funcionales.
- Resultado de los tests que se ejecutaron.
- Burndown chart.

## 13. Tecnologías

- Scala: para el desarrollo de los módulos cliente, núcleo e interfaz gráfica se utilizará el lenguaje de programación Scala. El mismo está diseñado teniendo en mente concurrencia, expresividad y escalabilidad. El resultado de la compilación produce código objeto que corre sobre la máquina virtual de Java, aprovechando la estabilidad y madurez de la misma. Provee un modelo de comunicación por mensajes basado en actores que simplifica y abstrae de los problemas clásicos que surgen de la naturaleza concurrente del dominio. Se eligió este lenguaje ya que ofrece una visión alternativa a los problemas clásicos de la concurrencia y será de soporte principalmente para el núcleo de la aplicación.
- Java: el lenguaje Scala permite utilizar código Java y viceversa. El plugin que se desarrollará para el IDE Eclipse será desarrollado con el lenguaje Java, ya que Eclipse se encuentra desarrollado en esa plataforma.
- Spring: framework para la inyección de dependencias.
- Maven: herramienta para la gestión y automatización de los procesos del ciclo de vida del proyecto en términos de compilación, dependencias, tests, integración, etc.
- Framework de unit testing: Scala Test y TestNG.
- Versionado GIT: sistema de versionado de código distribuido.

- Eclipse IDE: es el estándar de facto entre los entornos de desarrollo integrado para Java.
- GitHub: el código fuente será alojado en este sitio. El mismo provee gráficos y estadísticas útiles sobre el desarrollo del código fuente.

## 14. Infraestructura necesaria

Para la utilización del producto se requerirá la siguiente infraestructura de base. Se distinguirán 2 tipos de configuraciones según la modalidad en la cual se haga uso del producto. En primer lugar, para el caso del modo peer-to-peer solo será necesario disponer de las terminales o puestos de trabajo que deseen participar de una sesión de edición junto con una conexión de red que soporte el conjunto de protocolos TCP/IP entre las mismas. No es necesario un servidor central o dedicado.

Los requerimientos para correr el producto son equivalentes a aquellos necesario para correr un entorno de desarrollo convencional.

En la figura 3 se observa un posible escenario reflejando la arquitectura peer-to-peer:

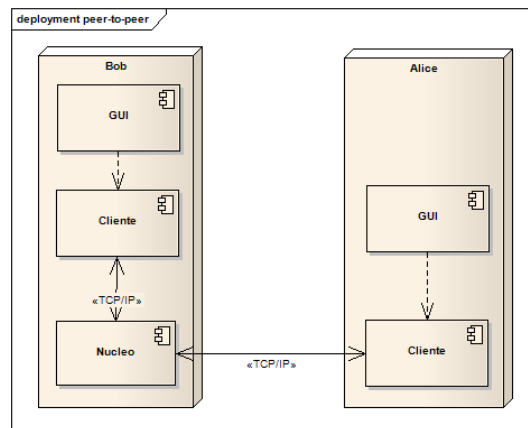


Figura 3: Peer-to-Peer

En el segundo caso, se dispone de una arquitectura cliente-servidor en la cual participa un servidor dedicado corriendo el núcleo de la aplicación y los distintos clientes que se conectan con el mismo. La conexión se establece entre el componente cliente y el componente núcleo abstrayendo a la aplicación GUI de la complejidad subyacente.

Se puede ver en la siguiente figura:

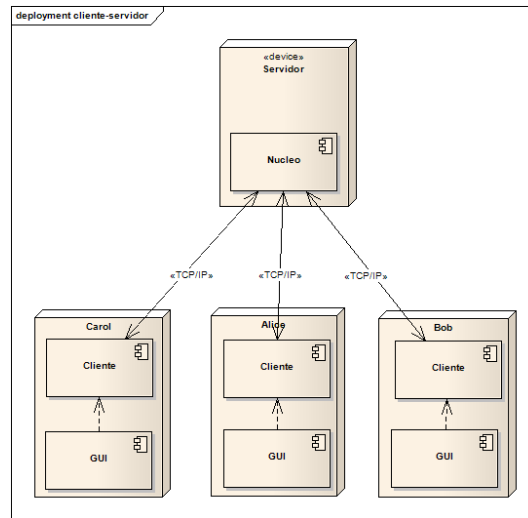


Figura 4: Cliente-Servidor

Esta alternativa permite concentrar todos los documentos que editan los usuarios en un solo punto.

En ambos casos no se necesita conectividad a internet.

## 15. Criterios de calidad

Se realizarán dos tipos de pruebas sobre el producto una vez iniciado el desarrollo. Como primer tipo se realizarán pruebas unitarias sobre el código fuente desarrollado. El objetivo es encontrar problemas en la lógica de la aplicación y a su vez documentar indirectamente el uso de la API.

El porcentaje del código fuente que se pretende cubrir con las pruebas unitarias es del 60 %. El foco de las pruebas se concentrará en los módulos núcleo y cliente del producto.

El segundo tipo de prueba a realizar son las pruebas funcionales. Estas tienen como característica que son difícilmente automatizables y deben ser realizadas por el usuario que utilizará la aplicación. Las pruebas se correrán al finalizar cada uno de los sprints definidos en la planificación de modo que funcionalidades ya desarrolladas no sean afectadas.

La definición de las pruebas a ejecutar se realizará al inicio de cada sprint.

## 16. Riesgos

A continuación se hará una lista de riesgos que fueron relevados al inicio del desarrollo del proyecto.

ID	Riesgo	P	I	E	Plan de mitigación
1	Demoras en el desarrollo de las funcionalidades por la poca experiencia en el desarrollo con Scala.	2	3	6	Retrasar el desarrollo para incorporar conocimientos sobre las herramientas. Cambiar la herramienta por otra en la que se tenga mayor experiencia.
2	Demoras en el desarrollo de la integración con Eclipse debido al desconocimiento de la arquitectura del mismo.	2	2	4	Integrar el producto con otro IDE. Incorporar conocimiento acerca de la arquitectura del Eclipse y su integración de plugins.
3	Demoras en el proyecto por la no aprobación de la propuesta de trabajo profesional.	1	3	3	Iniciar la construcción a partir de un subconjunto de requerimientos consensuados. Acordar qué falta definir. Evaluar la posibilidad de incluirlo dentro del proyecto o si es necesaria una ampliación.
4	Demoras en la definición de la arquitectura produce re-trabajo.	2	3	6	Realizar una definición temprana de la arquitectura y documentarla.

Los valores usados en las tablas corresponden a las siguientes definiciones:

Probabilidad (P): valor numérico entre 1 y 3.

Impacto (I): valor numérico entre 1 y 3.

Exposición (E):  $P * I$ .

## 17. Glosario

- **API:** una interfaz de programación de aplicaciones o API (del inglés application programming interface) es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Usados generalmente en las bibliotecas.
- **IDE:** entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.
- **Plugin:** un complemento es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.
- **WBS:** estructura de descomposición del trabajo o EDT, también conocido por su nombre en inglés Work Breakdown Structure o WBS, es una estructura exhaustiva, jerárquica y descendente formada por los entregables a realizar en un proyecto. La EDT es una herramienta muy común y crítica en la gestión de proyectos.
- **Diagrama de Gantt:** es una popular herramienta gráfica cuyo objetivo es mostrar el tiempo de dedicación previsto para diferentes tareas o actividades a lo largo de un tiempo total determinado. A pesar de que, en principio, el diagrama de Gantt no indica las relaciones existentes entre actividades, la posición de cada tarea a lo largo del tiempo hace que se puedan identificar dichas relaciones e interdependencias.
- **Burndown Chart:** es una representación gráfica del trabajo por hacer en un proyecto en el tiempo. Usualmente el trabajo remanente (o backlog) se muestra en el eje vertical y el tiempo en el eje horizontal. Es decir, el diagrama representa una serie temporal del trabajo pendiente. Este diagrama es útil para predecir cuándo se completará todo el trabajo. Usualmente se usa en el desarrollo ágil de software, especialmente con Scrum.

## Referencias

- [1] Ciancio, Gilioli, *Visión Trabajo Profesional*. Facultad de Ingeniería. Universidad de Buenos Aires.
- [2] Nichols, Curtis, Dixon and Lamping, *High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System*. Xerox PARC.
- [3] Ken Schwaber, *Agile Software Development with Scrum*. Prentice Hall, Octubre 2001.
- [4] *Google Docs*. Google Inc., <http://docs.google.com>.
- [5] *Google Wave*. <http://wave.google.com>.
- [6] Corvalius, *BeWeeVee*. <http://www.beweevee.com>.
- [7] Mustafa K. Isik, *COLA*. [http://wiki.eclipse.org/RT\\_Shared\\_Editing](http://wiki.eclipse.org/RT_Shared_Editing) .

# Trabajo Profesional: Parallel-Editor

Ciancio Alessio, Mauro Lucas  
Gilioli, Leandro Ezequiel

{maurociancio,legilioli}@gmail.com

2do cuatrimestre de 2010

## Resumen

En el presente trabajo se pretende encontrar y mostrar una solución al problema de la edición concurrente de documentos en tiempo real y distribuida. Este escenario se plantea en diversas áreas incluyendo el desarrollo de software. La técnica *pair-programming* [1], generalmente propuesta por las metodologías ágiles, puede verse beneficiada al usar el producto que se implementa en este trabajo profesional.

Se desarrolló una solución que permite la edición simultánea de uno o más documentos, asegurando que no exista divergencia entre las partes aún si la conexión entre ellos posee alta latencia. A su vez, se utiliza un mecanismo optimista para modificar los documentos, en el cual no se requiere de un árbitro central que coordine las partes involucradas.

Por último, la solución se integra con el IDE Eclipse [2] mediante un plugin desarrollado para este trabajo profesional facilitando la aplicación de la técnica antes mencionada.



# Índice

<b>1. Presentación</b>	<b>4</b>
<b>2. Motivación</b>	<b>5</b>
<b>3. Otras soluciones similares</b>	<b>7</b>
3.1. Google Docs [12] y Google Wave [13] . . . . .	7
3.2. BeWeeVee [14] . . . . .	7
3.3. COLA [15] - Eclipse Plugin . . . . .	7
3.4. Comparación con la presente propuesta . . . . .	8
<b>4. Justificación teórica de los algoritmos</b>	<b>9</b>
4.1. Definición de <i>xform</i> . . . . .	14
<b>5. Búsqueda de la solución</b>	<b>15</b>
<b>6. Elección de las tecnologías utilizadas</b>	<b>16</b>
6.1. Lenguaje de programación Scala . . . . .	16
6.1.1. Modelo de Actores . . . . .	17
6.2. IDE Eclipse . . . . .	18
6.3. Maven . . . . .	18
6.4. Spring Framework . . . . .	19
6.5. GIT . . . . .	19
6.6. JUnit [18] / EasyMock [19] / TestNG [20] . . . . .	19
<b>7. Arquitectura de la solución</b>	<b>21</b>
7.1. Módulos . . . . .	21
7.2. Despliegue de la solución . . . . .	22
7.2.1. Cliente-Servidor . . . . .	23
7.2.2. Peer-To-Peer . . . . .	23
7.3. Despliegue de actores en el proceso kernel . . . . .	25
7.4. Despliegue de actores en el proceso cliente . . . . .	27
7.5. Protocolo utilizado para enviar mensajes por la red . . . . .	28
7.6. Diagrama de clases del kernel . . . . .	28
7.7. Circuito de mensajes y operaciones . . . . .	29
7.8. Estado de Session y DocumentSession . . . . .	30
7.9. Cómo integrar esta solución a productos de terceros . . . . .	31
7.9.1. API's gráficas y Threads . . . . .	31
7.9.2. Editor de texto . . . . .	32
7.9.3. Implementación de la interfaz Documents . . . . .	33

<b>8. Riesgos</b>	<b>34</b>
8.1. ID 1: Demoras en el desarrollo de las funcionalidades por la poca experiencia con Scala . . . . .	34
8.2. ID 2: Demoras en el desarrollo de la integración con Eclipse debido al desconocimiento de la arquitectura del mismo . . . .	34
8.3. ID 3: Demoras en el proyecto por la no aprobación de la propuesta de trabajo profesional . . . . .	34
8.4. ID 4: Demoras en la definición de la arquitectura produce re-trabajo . . . . .	34
8.5. Riesgos no relevados . . . . .	35
<b>9. Problemas enfrentados durante el desarrollo</b>	<b>36</b>
<b>10. Metodología de desarrollo</b>	<b>37</b>
10.1. Sprint 1 . . . . .	37
10.2. Sprint 2 . . . . .	39
10.3. Sprint 3 . . . . .	40
10.4. Otras tareas . . . . .	41
<b>11. Futuras líneas de investigación</b>	<b>43</b>
<b>12. Conclusiones</b>	<b>45</b>
<b>13. Fuentes</b>	<b>46</b>
13.1. RemoteMessages.scala . . . . .	46
13.2. ClientMessages.scala . . . . .	49
13.3. DocumentMessages.scala . . . . .	49
13.4. SocketNetworkConnection.scala . . . . .	49
13.5. JupiterSynchronizer.scala . . . . .	50
13.6. BasicXFormStrategy.scala . . . . .	51
13.7. DocumentData.scala . . . . .	53
<b>14. Manual de Usuario</b>	<b>54</b>
14.1. Installation . . . . .	54
14.1.1. Installation within Eclipse . . . . .	54
14.1.2. Manual installation . . . . .	55
14.2. Usage . . . . .	56
14.2.1. Prerequisites . . . . .	56
14.2.2. I have it installed, now what? . . . . .	57
14.2.3. Connecting to other sessions . . . . .	58
14.2.4. Configuration . . . . .	58

# 1. Presentación

Este trabajo profesional fue desarrollado por:

- Ciancio Alessio, Mauro Lucas.  
Padrón: 86.357.  
maurociancio@gmail.com
- Gilioli, Leandro Ezequiel.  
Padrón: 86.075.  
legilioli@gmail.com

El profesor tutor de este trabajo profesional es el Lic. Pablo Cosso.

El nombre del proyecto es **Parallel-Editor** (en castellano **Editor Paralelo**) y el repositorio con la última *release* puede encontrarse en [21].

## 2. Motivación

El desarrollo de software en sus diversas tareas ha dejado de ser un trabajo puramente individual y requiere interacción de varias personas que en conjunto suman sus capacidades para lograr un producto de calidad superior.

Bajo esta forma de trabajo es necesario disponer de herramientas que ayuden a que los tiempos requeridos por la coordinación e interacción de los integrantes de un grupo de trabajo sean bajos de forma tal que el grupo sea productivo.

Con este objetivo, las herramientas utilizadas deben estar diseñadas e implementadas apuntando a integrarse con las metodologías y herramientas existentes sin representar un obstáculo para el usuario y en lo posible, que su costo sea relativamente bajo.

La idea del presente trabajo profesional nace de la necesidad de participar de una sesión de desarrollo, en la cual los participantes no están físicamente en el mismo lugar. En algunos casos en particular, esta tarea requiere un *feedback* instantáneo entre los participantes que con herramientas existentes no se puede ofrecer.

Estas herramientas no logran que el feedback sea lo suficientemente rápido dado que trabajan en un ambiente en el cual es necesario respetar un protocolo para la modificación del estado de un documento limitando la interactividad de la tarea de edición. Por ejemplo: un usuario que está trabajando sobre un determinado archivo debe realizar todos los cambios que quiere introducir, guardar la nueva versión del archivo modificado y enviarlo a sus colaboradores para que éstos estén al tanto de los cambios que introdujo (feedback). Mientras dura este proceso los colaboradores no pueden realizar cambios al documento y si lo hacen será necesario que realicen un proceso comúnmente llamado *merge*, de forma tal que el estado final del documento sea el mismo para cada uno de los participantes. En el caso en que aumenta el número de colaboradores, este proceso se hace lento y engorroso.

Este esquema funciona bien en los casos en los cuales es baja la concurrencia sobre un documento, es decir la edición de un mismo documento por parte de más de un usuario es ocasional o en períodos de tiempo disjuntos. Ejemplo: en el caso de que varios desarrolladores estén trabajando en un mismo proyecto con código fuente compartido, existen herramientas cuya efectividad está comprobada, como los sistemas de control de versiones (entre ellos SVN, GIT, Mercurial) o servidores de archivos compartidos.

La solución desarrollada en este trabajo profesional se pensó para resolver los problemas que surgen en este último caso, a saber:

- Distribución geográfica: no es necesario estar en la misma ubicación física para que el proceso de desarrollo sea eficiente.
- Necesidad de un proceso de merge: el proceso de merge es realizado por el software en cada sitio de edición garantizando que el estado final del documento es el mismo para todos los participantes. De esta manera se ahorra tiempo y se reducen los errores frecuentes o retrabajos los cuales son derivados de estos procesos.
- Alta latencia del feedback: los cambios en el estado del documento son reflejados en tiempo real para todos los participantes.

La solución hace uso de tecnologías y herramientas existentes para proporcionar las funcionalidades que resuelven los problemas antes descritos. Como ejemplo de esto, la solución se integró dentro de Eclipse, el entorno de desarrollo integrado de facto para el lenguaje de programación Java.

### 3. Otras soluciones similares

Durante la etapa de concepción del proyecto se analizaron otras soluciones similares al problema anteriormente explicado. La descripción de cada una de ellas junto con la comparación de las mismas respecto de la presente solución fue detallada en el documento “Propuesta de Trabajo Profesional” [4] y se muestra a continuación.

A modo de resumen se presenta el cuadro comparativo 1.

#### 3.1. Google Docs [12] y Google Wave [13]

Provee la posibilidad de editar en tiempo real concurrentemente documentos con formato utilizando un navegador compatible. Posibilita la participación de múltiples usuarios en línea.

Sin embargo, su mayor desventaja consiste en la dependencia de una conexión a internet para la disponibilidad del servicio. No es posible hasta ahora, la utilización del mismo en una red LAN privada.

Además es sólo utilizable a través de la interfaz web del navegador por lo cual su integración con herramientas de terceros no es posible. Cabe aclarar que Google ha abandonado el desarrollo de Google Wave.

#### 3.2. BeWeeVee [14]

Framework para la integración de funcionalidades de colaboración en tiempo real en aplicaciones. Se provee como un *software development kit* para el desarrollo sobre la plataforma .NET. Por este motivo, si bien abarca gran cantidad de desarrollos que hacen uso de dicha plataforma, no cubre totalmente el espacio de potenciales aplicaciones ya que las aplicaciones que no se desarrollan en .NET no pueden integrarlo.

Es de licencia libre para uso académico y aplicaciones *open source*, aunque tiene un costo para la integración en desarrollos privados.

#### 3.3. COLA [15] - Eclipse Plugin

COLA es un plugin para el IDE Eclipse que permite la colaboración en tiempo real de los usuarios para editar un mismo documento de código fuente. Está desarrollado en Java y está basado en el proyecto Eclipse Communication Framework (ECF). Su principal desventaja es la dependencia con el mismo y por otra parte, limita la cantidad de usuarios que pueden participar en una sesión de edición de código a dos participantes.

### 3.4. Comparación con la presente propuesta

El producto a desarrollar estará orientado a cubrir aquellos aspectos que las soluciones antes descriptas dejan de lado. Se pretenden liberar el código fuente bajo una licencia open-source, garantizar la portabilidad del mismo usando tecnologías que corren sobre la máquina virtual de Java, lograr independencia sobre otros frameworks o componentes de software y lograr que pueda haber más de dos participantes en la misma sesión de edición.

Solución	Características	Comparativa
Google Docs [12]	Edición de documentos en tiempo real desde un navegador.	Sólo puede utilizarse a través de un navegador e Internet. Código fuente cerrado.
Google Wave [13]	Comunicación y colaboración en tiempo real.	Ídem Google Docs. El proyecto ha sido abandonado por Google.
COLA [15] (ECF)	Integración con Eclipse para colaboración en tiempo real de código fuente.	Limita a dos usuarios la cantidad de participantes en una sesión. Depende del proyecto ECF.
BeWeeVee [14]	Framework para integración de funcionalidades de colaboración en tiempo real para la plataforma .NET.	Código fuente cerrado. Está desarrollado sólo para la plataforma .NET.

Cuadro 1: Tabla comparativa de soluciones

## 4. Justificación teórica de los algoritmos

El problema que resuelve este trabajo profesional se presenta genéricamente en escenarios en los cuales existen dos o más partes que interactúan sobre un modelo (ya sea un documento o cualquier otro tipo de objeto que almacene estado) aplicando operaciones sobre el mismo.

Cada operación modifica el estado del modelo que es común a todas las partes. La cantidad de tipos distintos de operaciones que se pueden aplicar sobre el modelo depende de la naturaleza del mismo. Por ejemplo, en un modelo que representa un documento de texto las operaciones pueden ser inserción o borrado de texto; en el caso de un modelo que represente un área de dibujo se podrán aplicar operaciones de dibujado de figuras, borrado, coloreado, etcétera.

La existencia de varios participantes compartiendo un modelo para su edición (sucesivas aplicaciones de operaciones) hace que sea posible que más de un participante pueda estar realizando operaciones sobre el mismo modelo. Al cabo de la aplicación de cada una de ellas todas las partes deben ver el mismo estado final, es decir, debe ser consistente. Por esta razón hay que tener en cuenta una manera de que esta condición se garantice.

El enfoque tradicional propone utilizar un modelo de sincronización centralizado en el cual para aplicar una operación sobre el modelo es necesario que el participante obtenga antes un control o autorización de un árbitro central. Una vez aplicada la operación, el estado resultante se transmite a todas las demás ubicaciones para que estén al tanto de los cambios generados como consecuencia de la aplicación de la operación remota. Si bien este método logra su cometido al preservar la consistencia del documento en todas las ubicaciones, la interactividad percibida en cada una de ellas es poco satisfactoria. Si al momento de querer aplicar una operación no se tiene la autorización o control del modelo, se deberá esperar para obtenerlo. Este tiempo de espera dependerá de la cantidad de participantes que se encuentran operando sobre el modelo en ese instante y del tiempo que cada operación remota tome en aplicarse (ejemplos de uso de este mecanismo son sistemas tradicionales de control de versiones con bloqueo de archivos).

Para evitar este inconveniente y dar la impresión de una aplicación instantánea de las operaciones en cada una de las ubicaciones de los participantes, el *modelo de la transformada operacional* (OT) [9] puede aplicarse. En éste, cada participante tiene una copia del modelo, sobre la cual aplica instantáneamente las operaciones que genera localmente. Luego de esto, notifica el cambio a todos los demás participantes que forman parte del proceso colaborativo. Al momento en que se reciben operaciones remotas en una ubicación, estas no son aplicadas, sino que se analiza el tipo de operación que se



recibió y la secuencia de operaciones que fueron aplicadas en esa ubicación anteriormente. A partir de esta información es posible transformar la operación original en otra que cumple la propiedad de que al aplicarse al modelo local garantiza que el estado resultante será el mismo que el de los modelos de todas las demás ubicaciones.

El algoritmo de sincronización que se implementó fue el algoritmo de **Júpiter** [6]. El proceso de implementación de este algoritmo se divide en varias etapas.

Como primer punto es necesario definir qué tipo de componentes se va a utilizar para representar el modelo de documento. En esta solución se trabajará exclusivamente con el componente denominado cuadro de texto (*textfield* en inglés). Este componente gráfico permite que se puede insertar texto en posiciones definidas por el usuario y existe en la mayoría de las bibliotecas de interfaces gráficas disponibles actualmente. Es posible considerar otros tipos de componentes que no se implementan en esta solución, como por ejemplo: una zona para dibujar entre varios usuarios, un slider, botones, checkboxes, etc.

Una vez que se definieron el o los componentes es necesario definir las operaciones que podrán ser aplicadas sobre los mismos. Éstas pueden ser definidas como las posibles modificaciones que se hacen sobre el componente. Por ejemplo, para el caso de cuadro de texto una posible operación es agregar el texto "hola" en la posición 10. Las operaciones necesarias para modificar el estado de un documento de texto plano en el componente utilizado en este trabajo son dos: agregar texto en una posición y borrar texto en una posición. Por razones que se explicarán más adelante, es deseable mantener la cantidad de operaciones distintas y la cantidad de parámetros de las mismas al mínimo.

Por último, una vez que se definieron los componentes a utilizar y las operaciones a realizar sobre los mismos, la implementación del algoritmo de sincronización que fue utilizado en esta solución requiere de una función de transformación de operaciones que cumple con ciertas características y se denomina *xform* [6].

Para comprender la importancia de la utilización de un algoritmo de sincronización y la transformación de operaciones realizadas por la función *xform* se procederá a plantear el siguiente escenario (y su descripción gráfica en la figura 1):

Suponemos que dos usuarios están compartiendo un documento colaborativamente. El estado del documento inicial en ambas ubicaciones es el mismo y corresponde a la palabra "HOLA". El primer caso que se plantea como ejem-

plo corresponde a modificaciones al documento de manera tal que las mismas no se superponen en el tiempo. En la ubicación 1 se inserta el carácter 'S' en la posición 4. Esta operación se aplica localmente y es transmitida a la ubicación número 2. En este instante la copia local del documento es la palabra "HOLAS". Mientras tanto en la ubicación 2 que posee la copia original del documento ("HOLA"), cuando se recibe la operación generada por la ubicación 1 se aplica localmente resultando ambas copias del documento idénticas. De igual forma en la ubicación 2 se genera otra operación (*borrar carácter H*) que se aplica localmente y luego es transmitida a la ubicación 1. En ésta se aplica la operación recibida y ambas ubicaciones terminan con el mismo estado del documento.

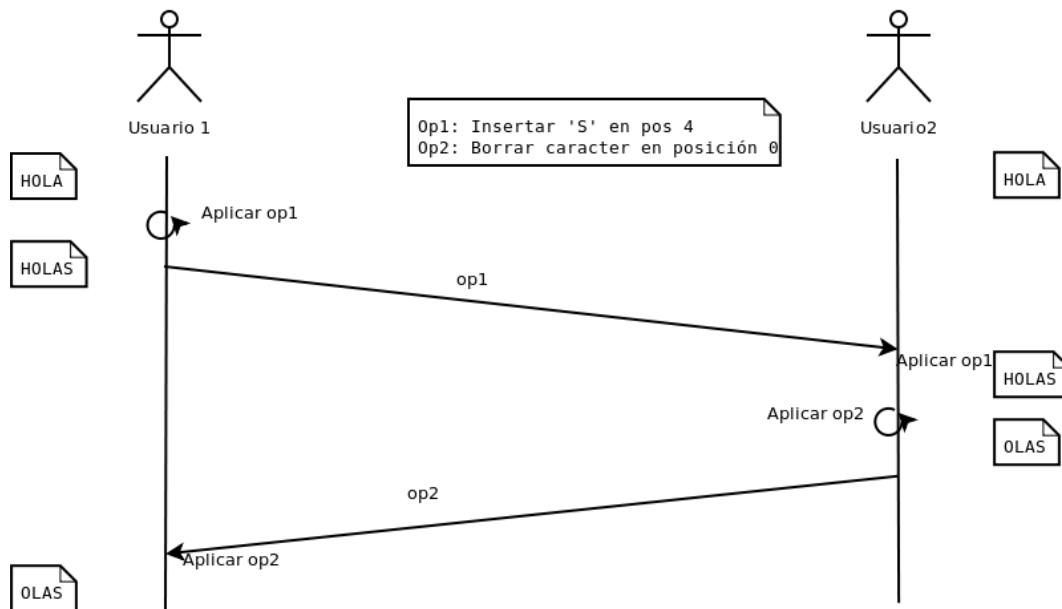


Figura 1: dos ubicaciones aplican operaciones sobre el modelo

En el caso presentado, no se observan problemas de sincronización debido a que la segunda operación fue generada cuando la primera ya había sido procesada en la ubicación 2.

Ahora, se presenta otro ejemplo en el que sí existe un problema de sincronización (representado en la figura 2). Las partes luego de procesar las operaciones tendrán documentos distintos. Esto sucede dado que la ubicación 2 genera una operación local sin aún haber recibido la operación generada por la otra parte.

Nuevamente, el estado inicial del documento en ambas ubicaciones es **HOLA**. En la ubicación 1 se generará la operación *insertar C en posición 0* llevando el documento al estado **CHOLA**. Antes de que se reciba la operación en la ubicación 2, el usuario en esta ubicación generará la operación de borrado del carácter O. La operación es transmitida como “*borrar carácter en posición 1*” ya que su estado actual es **HOLA**. Luego de aplicarse localmente la operación el estado del documento en el sitio 2 es **HLA**. En este momento ambas operaciones se encuentran viajando hacia las otras partes. En la ubicación 1 cuando recibe la operación generada remotamente, se aplica llevando al documento al estado **COLA** ya que la operación indicaba borrar un carácter en la posición 1. Luego, en la ubicación 2 es recibida la operación generada en el sitio 1. Al aplicarse localmente lleva el documento a un estado **CHLA**. Como se observa, en ambos sitios se obtiene un documento distinto al finalizar la sesión.

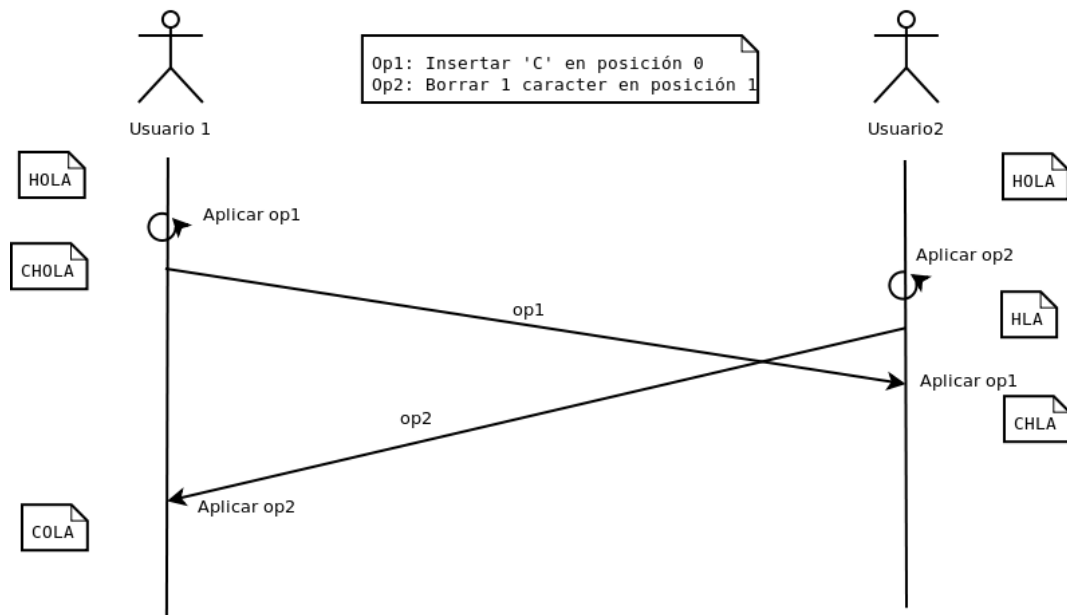


Figura 2: dos ubicaciones aplican operaciones sobre el modelo

La divergencia en el estado final del documento se produce por que se aplicaron las operaciones en cada ubicación sin haber sido transformadas previamente. En el caso de operaciones no conflictivas la función transformación de operaciones será la función identidad. Sin embargo, para los casos en los cuales las operaciones son conflictivas la operación resultante transformada será distinta a la original. Para este último caso la operación 2 recibida

desde la ubicación 2 debió haber sido corregida por la función de transformación y haber resultado en “*borrar carácter en posición 2*” (y no 1 como originalmente era la operación).

Presentada esta problemática es necesario definir una función  $xform$  con las siguientes propiedades:

$$xform(op1, op2) = \{op1', op2'\} \quad (1)$$

dónde  $op1$  es la operación generada por el usuario en la ubicación 1 y  $op2$  es la operación generada por el usuario en la ubicación 2. La aplicación de la función da como resultado otro par de operaciones  $op1'$  y  $op2'$  que cumplen con la propiedad de que si la ubicación 1 aplica  $op1$  seguida de  $op2'$  y si la ubicación 2 aplica  $op2$  seguida de  $op1'$ , entonces ambas ubicaciones terminarán con el mismo estado del documento. Esto requiere que  $op1$  y  $op2$  hayan sido generadas a partir del mismo estado del documento.

Para el caso anterior, se mostrará a continuación la función  $xform$ . Se parte del estado inicial del documento **HOLA**.

En la ubicación 1 se aplica  $op1$  y al momento en el que se recibe  $op2$  se la transforma para obtener la operación que debe aplicarse. A su vez, en la ubicación 2 se procede de forma similar aplicando primero localmente  $op2$  y transformando  $op1$  cuando se recibe.

En ambas ubicaciones las operaciones transformadas se obtienen de la aplicación de  $xform$ :

$$\begin{aligned} xform(&Insertar\ Caracter\ C\ en\ Pos = 0, \\ &Borrar\ Caracter\ en\ Pos = 1) = \\ &\{Insertar\ Caracter\ C\ en\ Pos = 0, \\ &Borrar\ Caracter\ en\ Pos = 2\} \end{aligned}$$

De aquí resulta que  $op1'$  es *Insertar Carácter C en Pos = 0* y  $op2'$  es *Borrar Carácter en Pos = 2*.

$op1'$  resultó ser igual a  $op1$  mientras que  $op2'$  fue desplazada con respecto a  $op2$ .

En la ubicación 1 se aplica  $op2'$  sobre el **CHOLA** resultando en: **CHLA**.

En la ubicación 2 se aplica  $op1'$  sobre **HLA** resultando en **CHLA**.

Nótese que aplicando las operaciones transformadas se logra la convergencia del estado del documento en las dos ubicaciones.

## 4.1. Definición de *xform*

La definición de esta función puede resultar compleja si se toma en cuenta la cantidad de combinaciones de las operaciones aplicables al modelo. Por esta razón se debe mantener al mínimo la cantidad de operaciones para simplificar el desarrollo de la función.

La función de transformación debe contemplar los casos en las que las operaciones sean conflictivas entre sí. Dos operaciones *op1* y *op2* son conflictivas si el estado final del documento al aplicar primero *op1* y luego *op2* es distinto al que se obtiene primero *op2* y luego *op1*. En el caso en que las operaciones no son conflictivas la transformación de las mismas es la transformación identidad.

La implementación de la función *xform* en la presente solución puede observarse en la clase `BasicXFormStrategy`. La misma fue basada en los papers de Júpiter e INRIA [5].

## 5. Búsqueda de la solución

El paper Júpiter [6] fue la base para la implementación del algoritmo de sincronización. Este paper fue encontrado durante el desarrollo de la visión [3] del producto. Otros papers están basados en éste y corrigen y proponen nuevas soluciones o correcciones.

Se detectaron problemas al utilizar operaciones de longitud mayor a uno (enfoque original que se le dio a la implementación). En un primer momento se detectaron problemas al utilizar operaciones de borrado de más de un carácter. Luego surgieron problemas al usar inserciones de más de un carácter. Para solucionar este problema se convirtieron las operaciones a operaciones de un carácter.

Este cambio no posee efectos colaterales importantes. Lo que se debe tener en cuenta es convertir las operaciones generadas por la aplicación cliente (o API gráfica) a operaciones de un carácter.

Durante el desarrollo de las pruebas se encontró un caso denominado Puzzle [5] que producía una falla en la sincronización. Se denomina Puzzle a una combinación de estados y operaciones producidas en determinadas ubicaciones que provoca que el algoritmo no garantice la convergencia del estado del documento para todos los usuarios. El paper de INRIA [5] propone una solución a este problema que demuestra formalmente las propiedades del algoritmo de sincronización utilizado.

Se implementaron los cambios propuestos por este paper junto con los tests para probar el correcto funcionamiento.

## 6. Elección de las tecnologías utilizadas

### 6.1. Lenguaje de programación Scala

En retrospectiva la elección de este lenguaje para el desarrollo de los principales módulos de la solución fue acertada. El principal beneficio que se obtuvo fue una gran simplificación al utilizar un modelo de concurrencia basado en actores [7, 8] que se detalla en la siguiente sección. Este modelo abstrae al desarrollador de la complejidad de utilizar mecanismos de sincronización y comunicación entre hilos como se realiza tradicionalmente en Java.

Por otro lado, es un lenguaje multiparadigma (Orientado a Objetos y Funcional). La ventaja de esto es que permite utilizar construcciones de lenguajes formales que resultan más concisas para resolver determinadas problemáticas.

El lenguaje pone el foco en la inmutabilidad de los objetos haciéndolo ideal para trabajar en ambientes concurrentes. La sintaxis del lenguaje Scala resultó altamente expresiva y concisa. La proporción final de código fuente del trabajo entre Scala y Java es la siguiente:

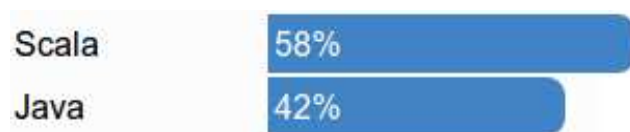


Figura 3: Porcentaje de código fuente según lenguaje de programación

El lenguaje Scala es compilado a bytecode de la Java Virtual Machine. El beneficio de esto es que el código fuente compilado de Scala es indistinguible con respecto al de Java permitiendo que los dos puedan correr en conjunto sin problemas de compatibilidad. No existe la necesidad de codificar nuevamente bibliotecas de Java ya que éstas pueden ser utilizadas desde Scala sin inconveniente alguno.

Teniendo en cuenta que el plugin para el IDE Eclipse fue lo único que se desarrolló en el lenguaje Java puede notarse que comparado con éste, el código Scala es altamente denso, es decir que permite lograr gran funcionalidad con pocas líneas de código. El plugin fue implementado en Java debido a que el Eclipse está implementando en este lenguaje.

Los módulos Cliente, Kernel, GUI, Common y Server fueron totalmente desarrollados en Scala.

La compatibilidad es total entre estos dos lenguajes en los dos sentidos

debido a lo explicado anteriormente con respecto al bytecode. Este punto fue clave para el desarrollo del plugin y para la elección de Scala como lenguaje base. La misma resulto de gran utilidad y aceleró el proceso de desarrollo.

### 6.1.1. Modelo de Actores

El modelo de actores implementado en el lenguaje de programación Scala ofrece una abstracción mucho mayor que la provista por lenguajes como Java. Libera al desarrollador de problemas de infraestructura tales como sincronización, locks, objetos monitores, etc.

Este modelo se basa en una entidad denominada Actor la cual es capaz de enviar y recibir mensajes de otros actores. Puede tomar decisiones en base a los mensajes que recibe y enviar respuestas a peticiones de servicios.

La comunicación entre actores se da mediante el uso de un buzón en el cual se encolan los mensajes que un actor debe procesar. Los mensajes se envían asincrónicamente, por lo cual un actor que desea enviar un mensaje simplemente lo envía y sigue con sus tareas. Luego, en el momento que desee, puede revisar su buzón para procesar mensajes si es que existe alguno.

Los actores en Scala son generalmente soportados mediante un pool de threads. Cuando un actor está en condiciones de ejecutar una porción de código es invocado para que lo haga. Cuando éste ha terminado de realizar el procesamiento notifica al framework para que le entregue el siguiente mensaje que debe procesar o que le de la posibilidad a otro actor para que haga sus tareas.

Los actores pueden ser vistos como threads implementados en el espacio de usuario del proceso en el que corren. El sistema operativo no es capaz de determinar cuales ni cuantos actores están corriendo sino que sólo posee información de los threads. Por lo tanto, como los actores están compartiendo threads éstos deben comportarse equitativamente para darle la posibilidad a los demás de ejecutar su código.

Para que se conserve la propiedad de equidad los actores deben tratar de evitar las siguientes situaciones:

- Procesamiento input/output sincrónico: no es recomendable que los actores hagan tareas costosas en tiempo como lo son el input/output sincrónicas. Por ejemplo: un actor leyendo un archivo desde disco.

Una posible alternativa es utilizar input/output asincrónico ya que es compatible con el modelo de actores.

- Uso de sockets: generalmente el uso de sockets es bloqueante por lo que no se recomienda hacer estas tareas en un actor.



- Procesamiento costoso: las tareas que demanden grandes tiempos de procesamiento (CPU-intensivas) tampoco son recomendables para realizar en un actor.

Si no se tienen en cuenta estos puntos se puede ocasionar que el resto de los actores no se ejecuten nunca.

Otro beneficio provisto por este modelo es que se pueden crear grandes cantidades de actores en comparación con threads. Cada thread necesita espacio para su stack (el cual puede ser del orden de 1MB) mientras que los actores comparten el stack del thread en el que corren. El actor es un objeto más y generalmente su tamaño es del orden de 1KB [31].

## 6.2. IDE Eclipse

Es el entorno integrado de desarrollo de facto para el lenguaje de programación Java y otros. La arquitectura está basada en plugins y está concebida para que la extensión de funcionalidad sea realizada por medio de ellos. Posee una gran cantidad de plugins desarrollados y una sólida documentación de la API.

Lo que se logró en el desarrollo del plugin del presente trabajo es que el mismo pueda integrarse con el IDE y agregarle las funcionalidades sin interferir con plugins existentes. El desarrollo es compatible con todas las funcionalidades incorporadas a los editores de texto. Por ejemplo: coloreo de código fuente, refactor, indentado, formateo, generación de código fuente.

Por otro lado, Eclipse provee un proceso de despliegue e instalación de plugins sencillo. Estos son publicados en Internet en cualquier servidor web y la instalación de los mismos se realiza apuntando a la URL del mismo.

## 6.3. Maven

Es una herramienta que se utiliza para controlar el ciclo de vida del proyecto en términos de compilación, testing, resolución de dependencias y despliegue. Provee una serie de tareas que facilitan estos procedimientos que deben repetirse continuamente en el proceso de desarrollo de un producto.

El principal beneficio de utilizar Maven es que permite hacer portable el entorno de desarrollo a nuevas estaciones de trabajo sin un esfuerzo extra. Una vez que el proyecto está configurado correctamente (mediante la declaración en el archivo `pom.xml`) pueden instalarse nuevos puestos de desarrollo de forma muy sencilla. Ésto es posible ya que las bibliotecas de las que un

proyecto depende se encuentran en Internet (o en una LAN) y pueden descargarse desde cualquier ubicación.

Su arquitectura basada en plugins permite que terceros agreguen funcionalidades que son publicados en los repositorios de Maven.

Mediante un plugin es posible integrar Maven con proyectos desarrollados en el lenguaje Scala.

## 6.4. Spring Framework

Es un framework de código abierto para el desarrollo de aplicaciones para la plataforma Java. Spring provee soporte para simplificar el desarrollo de diversas partes de aplicaciones enterprise. Generalmente es utilizado en la capa de servicios para la implementación de tareas como: logging, transacciones, seguridad, etc; y también utilizado para el acceso a los datos.

Además, ofrece un potente mecanismo de inyección de dependencias totalmente configurable mediante archivos XML, que permiten cambiar el comportamiento de un aplicación sin la necesidad de recompilar la misma.

Fue utilizado tanto el módulo GUI como el módulo Server. Solamente fue utilizado el mecanismo de inyección de dependencias.

Scala es totalmente compatible con Spring.

## 6.5. GIT

GIT es un sistema de control de versiones distribuido. No requiere de un servidor central para funcionar y permite trabajar aún si no se posee conexión de red.

Ofrece herramientas muy poderosas para la creación de ramas de desarrollo y merge de las mismas, generación de tags, etc.

A diferencia de los sistemas de control de versiones centralizados tales como SVN o CVS, GIT almacena toda la historia del proyecto en cada working-copy. El beneficio de esto es que la mayoría de las operaciones que se ejecutan sobre el repositorio no requieren del uso de la red y su aplicación es casi instantánea.

GIT fue utilizado para el desarrollo del proyecto junto con GitHub [21] a modo de repositorio. GitHub es un servicio gratuito que ofrece herramientas en la web para ver los *diff* de commits, hacer comentarios de los mismos, etc.

## 6.6. JUnit [18] / EasyMock [19] / TestNG [20]

Estos frameworks facilitan la creación de tests unitarios automatizados. Se utilizaron para verificar la correcta implementación de las funcionalidades

del producto.

Para realizar el testing de funcionalidades desarrolladas en Scala se utilizó el framework JUnit. Para Java se utilizó el framework TestNG.

EasyMock facilita la creación de *mock objects* [28] que son útiles para aislar partes que no se desean testear. Fue utilizado tanto en el desarrollo con Scala y con Java.

## 7. Arquitectura de la solución

### 7.1. Módulos

Como primer punto se mostrarán los componentes que conforman la solución. Está compuesto por 3 módulos principales:

- **Kernel:** el módulo kernel provee los servicios de colaboración en tiempo real de documentos. En él se encuentran las entidades que manejan el estado de los documentos, los participantes conectados al kernel y qué participantes se encuentran editando qué documentos.

Los documentos almacenan su estado actual de modo que pueda ser enviado a los nuevos participantes que ingresen a la sesión de desarrollo.

Los participantes pueden comunicarse entre ellos por un canal distinto al del documento, de modo que puedan organizarse para colaborar en la edición del documento. Este servicio es denominado chat y es implementando en este módulo.

Además, el mismo es capaz de determinar cuando un usuario se desconectó. Cuando este evento ocurre el participante es desuscripto de los documentos en los cuales estaba colaborando para que no interfiera con la sesión de edición.

- **Cliente:** define la API mediante la cual un cliente se conecta a un servidor colaborativo y edita documentos en tiempo real.

La API permite crear conexiones a múltiples servidores de colaboración simultáneamente y participar en varias sesiones de edición en el mismo instante de tiempo.

A su vez, este módulo publica las interfaces que los clientes del mismo deben implementar si se desea hacer uso de sus funcionalidades.

Este es el módulo que utilizan las aplicaciones de terceros para incorporar las funcionalidades de edición colaborativa y es incluido tanto por el módulo GUI como por el módulo Eclipse.

- **Common:** este módulo es utilizado por los dos módulos anteriores para compartir tipos y clases. No se espera que las aplicaciones clientes lo incluyan, sino que lo hagan a través de los módulos kernel y/o cliente. En este módulo se encuentran los mensajes que utilizan los clientes y el kernel para comunicarse entre sí.

Además, se encuentra la estrategia de sincronización que se utiliza para mantener el estado consistente de los documentos que poseen los participantes.

Existen además los siguientes módulos construidos en base a los anteriores:

- **Server:** es una aplicación que crea un servicio para compartir documentos que se quieren editar colaborativamente a través de una red TCP/IP.

Este módulo puede utilizarse si se desea un servicio de colaboración dedicado.

- **GUI:** es una aplicación cliente implementada usando Swing que ofrece la funcionalidad para conectarse a un servicio remoto y editar documentos existentes. Ofrece la posibilidad de compartir y editar documentos en un servidor colaborativo.

Provee al usuario de una interfaz gráfica en la cual se visualizan los documentos en edición y que permite insertar y eliminar texto de los mismos.

- **Eclipse Plugin:** integra la funcionalidades del módulo cliente y kernel dentro del IDE Eclipse. Hace uso tanto del módulo cliente (para conectarse a servidores de colaboración existentes) como del módulo kernel (para la creación de servidores de colaboración locales).

Se integra como plugin y permite la edición de archivos de texto y de código fuente dentro del IDE aprovechando todas las funcionalidades ya desarrolladas por la comunidad.

Las dependencias entre los módulos pueden ser observadas en la figura 4.

## 7.2. Despliegue de la solución

Existen dos esquemas de despliegue para la solución.

- Cliente-Servidor
- Peer-To-Peer

A continuación se describen las características y ventajas de cada esquema:

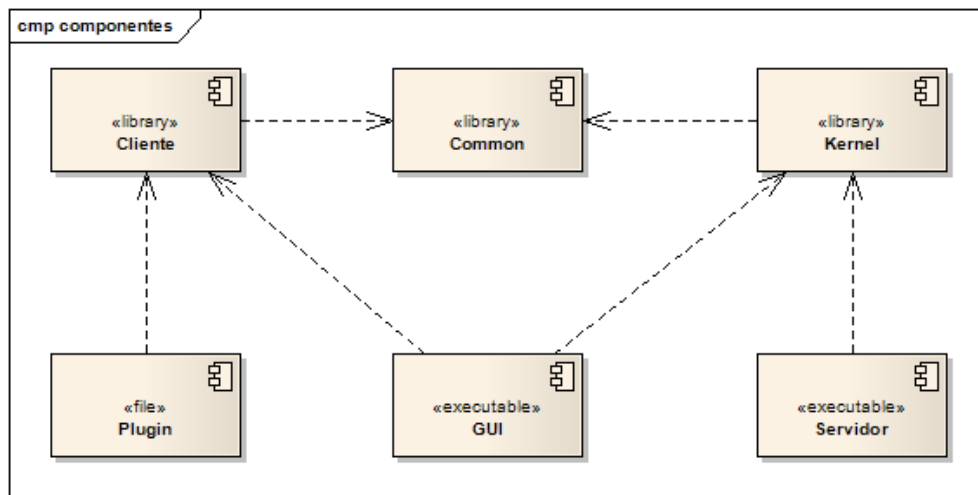


Figura 4: Componentes pertenecientes a la solución y sus dependencias

### 7.2.1. Cliente-Servidor

El primer caso consiste en disponer de un servidor dedicado para la gestión de documentos compartidos. El mismo recibe peticiones de N clientes que mediante una suscripción a los documentos pueden comenzar a realizar operaciones sobre los mismos.

El servidor es el encargado de reflejar los cambios introducidos por un cliente en los restantes.

Un diagrama de esta configuración puede observarse en la figura 5.

Los procesos involucrados en este tipo de despliegue son:

- Proceso servidor: en este proceso se encuentra ejecutándose los siguientes módulos: Kernel, Common y Server. Sólo hay una instancia de este proceso.
- Proceso cliente: en este proceso se encuentra ejecutándose los siguientes módulos: Cliente, Common y la aplicación correspondiente que puede ser GUI o Eclipse. La cantidad de instancias de este proceso dependen de la cantidad de usuarios que haya conectados al servidor de colaboración.

### 7.2.2. Peer-To-Peer

La segunda configuración para el despliegue de la solución consiste en no disponer de un servidor dedicado, sino que el mismo es ejecutado por uno

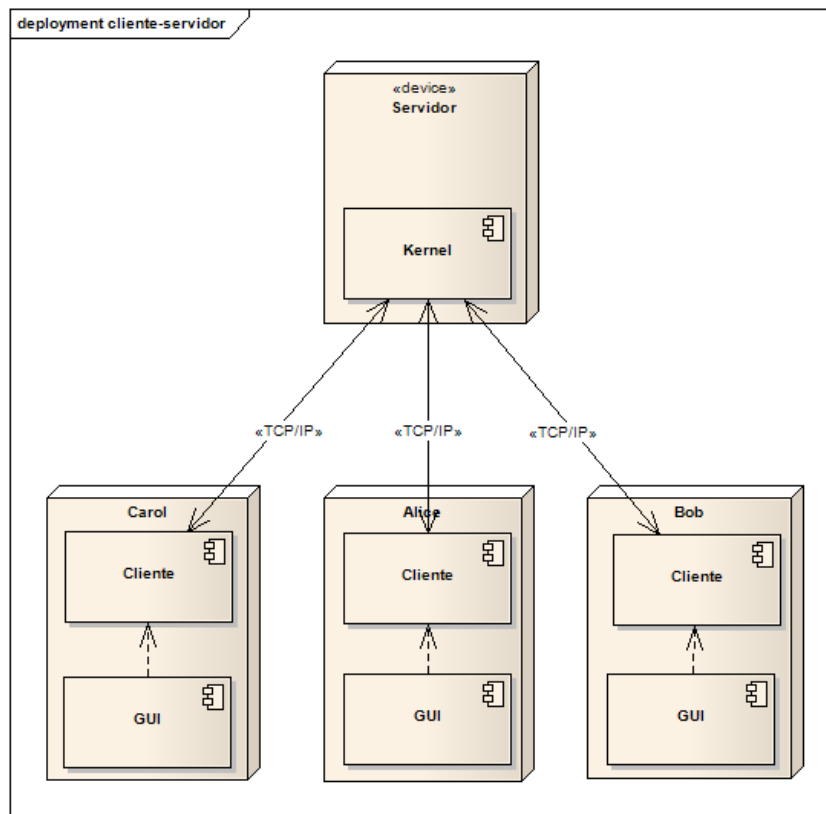


Figura 5: Despliegue usando esquema cliente-servidor

de lo pares. Este enfoque es el que se utiliza en el plugin, ya que el mismo permite crear un servicio a partir de un archivo que se quiere compartir.

Esta configuración es la más sencilla para un uso ocasional del servicio o cuando la participantes en la sesión de edición no es elevada. Las ventajas de esta configuración es que no se necesita un equipo central.

Si los participantes se encuentran en la misma LAN, no se requiere ninguna configuración extra. Si la sesión quiere realizarse a través de internet es necesario tener en cuenta que haya conectividad entre las partes (configuración de puertos y firewall).

Al momento de escribir esta documentación no es posible utilizar el servicio detrás de un proxy ya que los mismos filtran todo el tráfico que no sea HTTP. En cambio si es posible hacerlo utilizarlo usando una VPN.

Los procesos involucrados en esta configuración son:

- Proceso cliente con servicio de colaboración: uno de los participantes

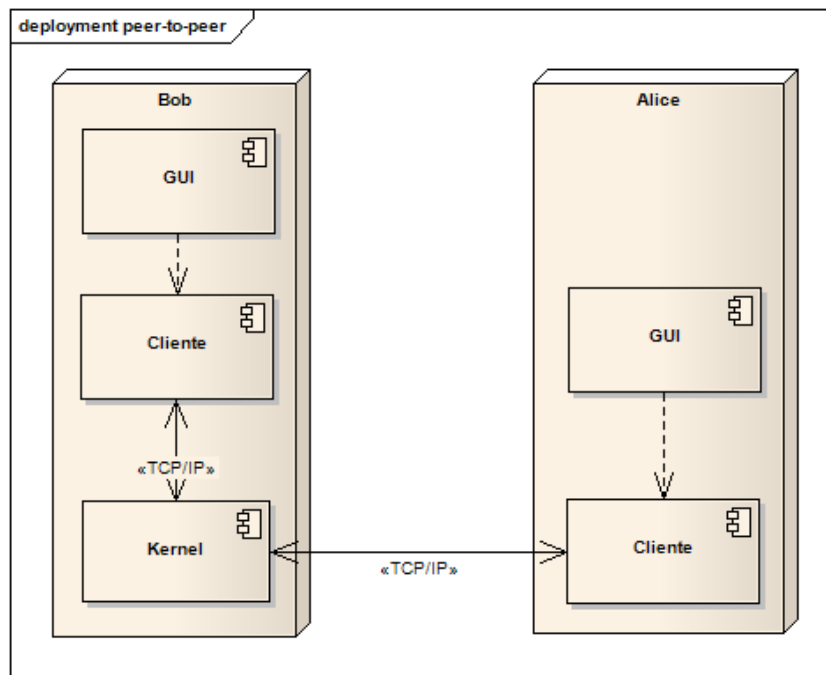


Figura 6: Despliegue usando esquema peer-to-peer

involucrados será el encargado de crear el servicio de colaboración. En éste se ejecutarán los siguientes módulos: Cliente, Kernel, Common y la aplicación que corresponda: GUI o Eclipse Plugin. Un solo proceso existirá de este tipo.

- Procesos cliente sin servicio de colaboración: los restantes participantes se comunicarán con el cliente que posee el servicio de colaboración. Los módulos que se ejecutarán en este tipo de cliente son: Cliente y Common. La cantidad de estos procesos dependerá de la cantidad de usuarios conectados al servicio de colaboración.

Este esquema puede observarse en la figura 6.

### 7.3. Despliegue de actores en el proceso kernel

En la figura 7 se observan los actores desplegados en el kernel.

Los actores fueron utilizados para cumplir el rol de intermediario ante el acceso a un recurso, serializando los accesos de modo de que no se produzcan conflictos de naturaleza concurrente.



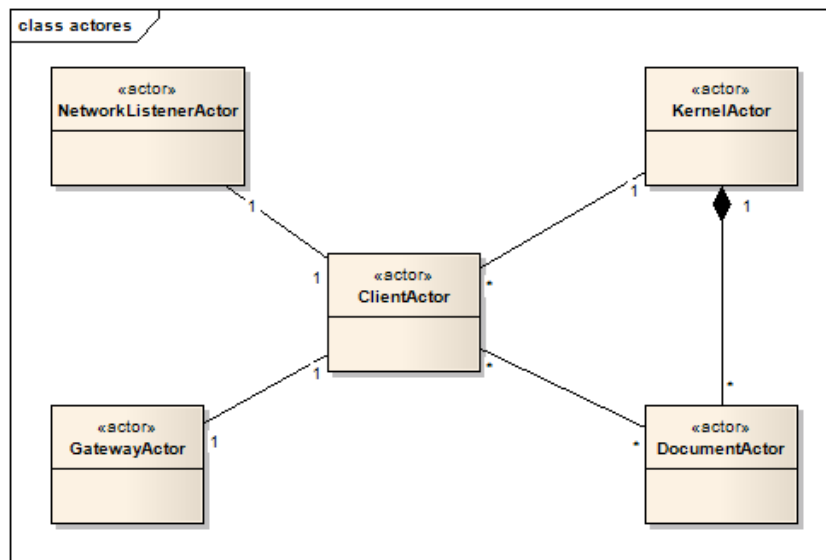


Figura 7: Despliegue de actores en el módulo kernel

Los principales actores del módulo kernel son los siguientes:

- **Kernel Actor:** actúa como intermediario del objeto kernel. La clase kernel gestiona las sesiones de los usuarios conectados y los documentos disponibles.
- **Document Actor:** actúa como intermediario de un objeto Document aplicando operaciones de los clientes sobre el mismo y replicándolas hacia los clientes restantes. Mantiene el estado del documento de modo que pueda servirlo a nuevas sesiones.
- **Client Actor:** es la representación de un cliente remoto frente al kernel. Se encarga de recibir mensajes del cliente remoto, enviarlos al kernel y enviar la respuesta al mismo. Los mensajes que se reciben de la red son capturados por el actor NetworkListener mientras que los mensajes que se deben enviar al cliente remoto son enviados por el GatewayActor.

Existe un protocolo interno representado por los mensajes que se envían entre sí. Los mensajes que conforman este protocolo pueden verse en el archivo: `DocumentMessages.scala`.

## 7.4. Despliegue de actores en el proceso cliente

En la figura 8 se observan los actores que se ejecutan en las aplicaciones que incorporan el módulo cliente.

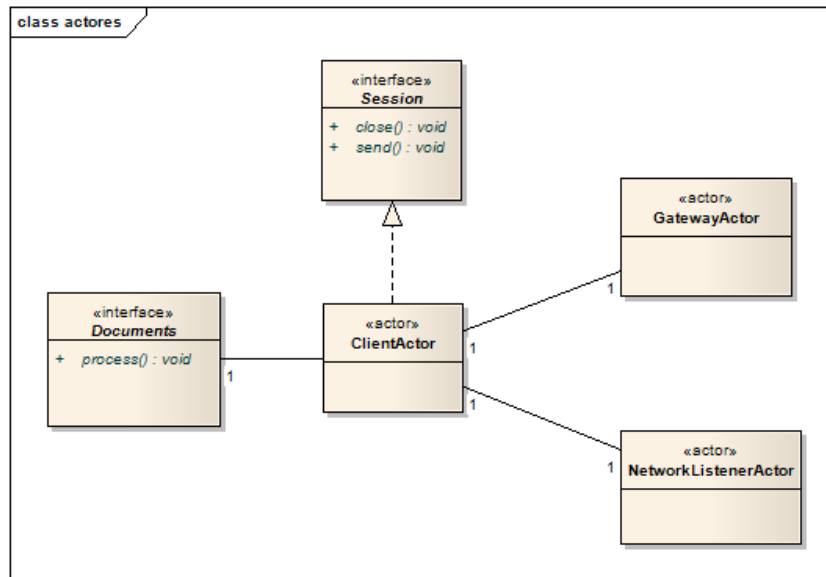


Figura 8: Despliegue de actores en el módulo cliente

Como se observa, el esquema es similar al encontrado en el kernel. Existe un Actor denominado `ClientActor` que es el encargado de coordinar los mensajes que deben enviarse hacia el kernel y de recibir las respuestas del mismo. Los actores `NetworkListenerActor` y `GatewayActor` son los encargados de leer y escribir mensajes por la red, respectivamente. En el módulo cliente se define una interfaz llamada `Documents` que es la que debe implementar la aplicación cliente para poder utilizar los servicios provistos por la API. A través de esta interfaz los mensajes son propagados hacia la aplicación cliente. Es responsabilidad que la aplicación cliente responda a estos mensajes.

Los mensajes que son enviados a la aplicación cliente pueden encontrarse en el archivo `ClientMessages.scala`. Con respecto a la cardinalidad de estos actores, podemos notar lo siguiente: por cada conexión que se realiza a un servidor de colaboración es creada una instancia de cada uno de estos actores.

Cabe aclarar que estos actores se encuentran detrás de un facade, para evitar que la aplicación cliente observe la complejidad subyacente. Este facade se denomina `Session` cuando es utilizado desde Scala y `JSession` cuando es utilizado desde Java. Se hizo esta diferencia para que los nombres de métodos sean legibles desde Java. (Nombres de métodos que son ilegales en Java son

legales en Scala y al ser compilados a bytecode son convertidos resultando en nombres no convencionales. Ejemplo: el nombre de método ! es convertido a \$bang).

## 7.5. Protocolo utilizado para enviar mensajes por la red

Los mensajes son serializados e hidratados utilizando el mecanismo de serialización de objetos de la JVM. Los mensajes que se envían por la red se encuentran en el archivo `RemoteMessages.scala`. Las clases que necesitan ser serializadas se encuentran marcadas con la anotación `@serializable`.

Se tuvo en cuenta durante la etapa de diseño que el protocolo sea fácilmente reemplazable por otro. A continuación listamos algunas posibles razones por la cual se podría reemplazar el protocolo por otro:

- Seguridad: agregar una capa que implemente cifrado de datos.
- Compresión: compresión de los mensajes para reducir la carga de la red.

En el archivo `SocketNetworkConnection.scala` se implementa el protocolo actual.

## 7.6. Diagrama de clases del kernel

En la figura 9 se observan las clases principales que trabajan en el kernel. Por simplicidad no se incluyen métodos de determinadas clases.

En el mismo se muestra qué interfaces son utilizadas por los clientes. Los clientes son los usuarios que se conectan remotamente al servicio de colaboración. Por lo tanto, el actor Cliente mostrado en el diagrama es un actor remoto y se ocupa el rol de intermediario entre el kernel y el cliente real que se encuentra del otro lado de la conexión.

Las interfaces que utiliza son las siguientes:

- Kernel: utiliza esta interfaz para pedir servicios al kernel de login, servicios de suscripción a documentos, servicios de chat, etc.
- Session: esta interfaz representa una sesión con el kernel. Todas las operaciones sobre el kernel requieren que se esté logueado al mismo. Esta interfaz ofrece la posibilidad de instalar un callback por la cual

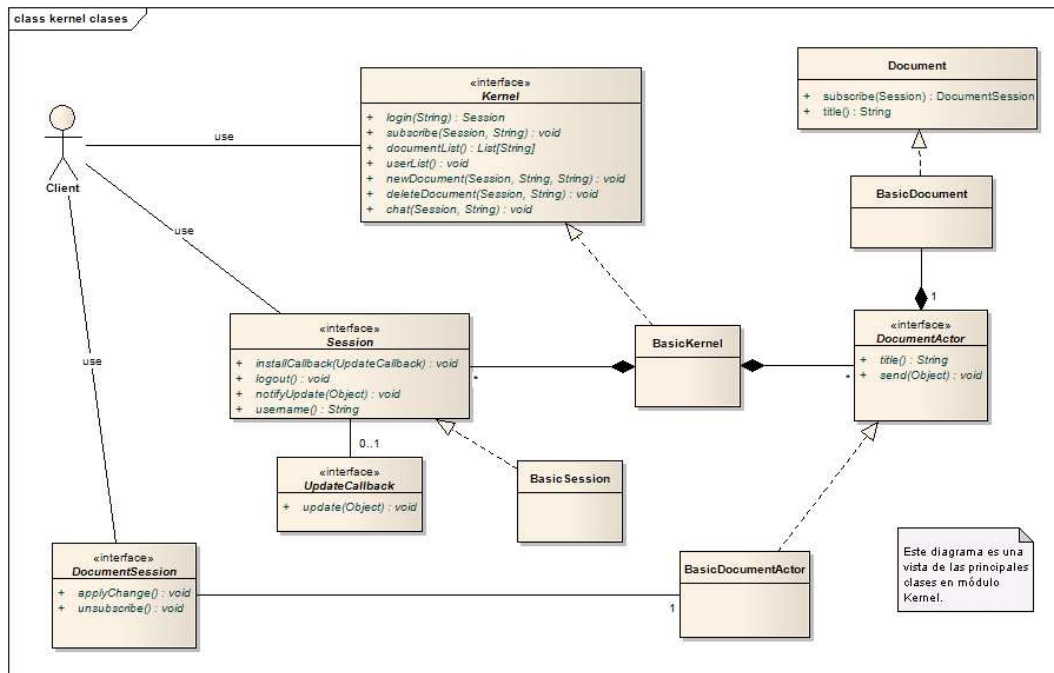


Figura 9: Clases principales en el módulo kernel

será notificada de mensajes provenientes del kernel. Entre estos mensajes podemos encontrar: mensajes de chat de otro usuario, mensajes de actualización de estado de un documento, etc. Es responsabilidad del cliente de instalar una callback para poder escuchar estos mensajes.

- Document Session: representa la sesión de edición de un documento en particular. Mediante esta interfaz es posible comunicarse con el documento para poder enviarle modificaciones.

## 7.7. Circuito de mensajes y operaciones

La figura 10 muestra la diferencia entre mensajes y operaciones.

Notar que se reciben mensajes del kernel y sobre los widgets (o componentes) se deben aplicar operaciones. Por ese motivo es necesario utilizar la misma estrategia de sincronización que se utiliza en el kernel. Las clases que implementan la sincronización se encuentran en los archivos `JupiterSynchronizer.scala` y `BasicXFormStrategy.scala`.

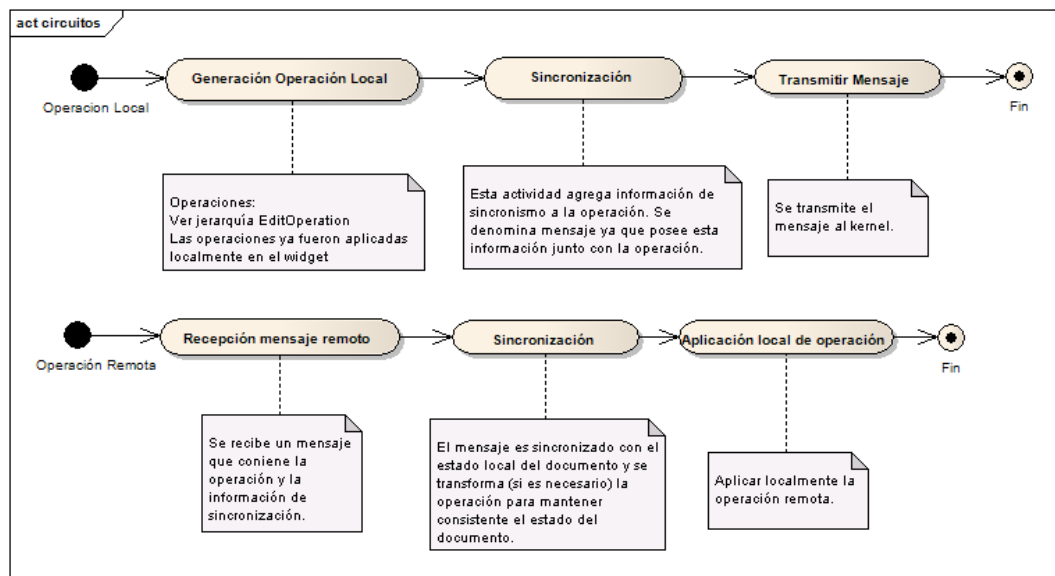


Figura 10: Circuito de mensajes y operaciones

## 7.8. Estado de Session y DocumentSession

En la figura 11 se muestra el diagrama de estados para el login de un usuario al kernel.

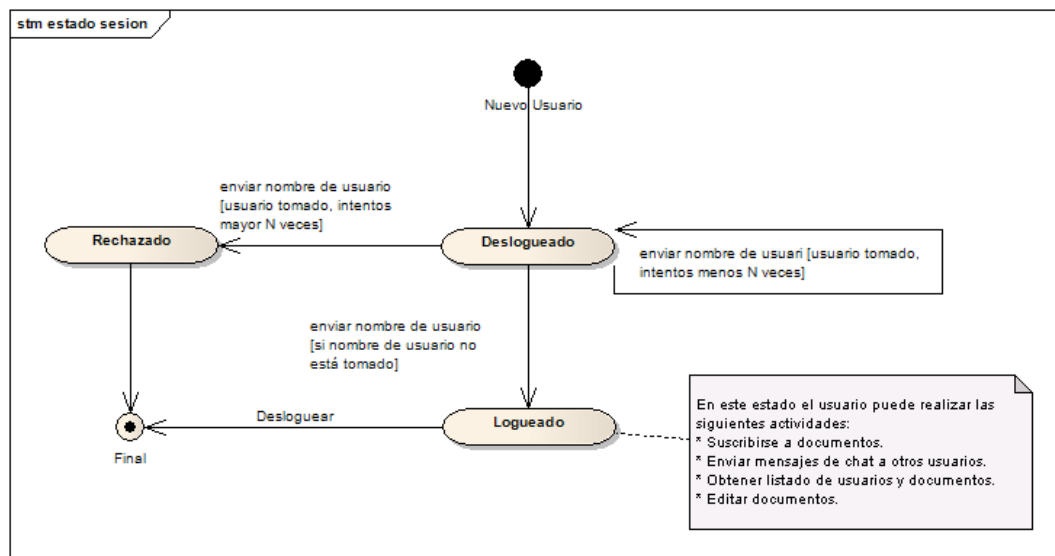


Figura 11: Diagrama de estados de la sesión del usuario

Con respecto a la `DocumentSession`, su diagrama de estados es muy sencillo. Solamente se observan dos estados, suscrito al documento y no suscrito. Únicamente cuando se encuentra en el estado suscrito es posible aplicar operaciones sobre el mismo. Si se trata de modificar el estado del documento cuando no se está suscrito al mismo, la operación será ignorada.

## **7.9. Cómo integrar esta solución a productos de terceros**

Para completar la documentación sobre la arquitectura de la solución se explicará cómo integrar este producto en otros ya existentes. Se mostrará que no es necesario rediseñar todo el producto para que pueda trabajar con `Parallel-Editor`.

Observaciones a tener en cuenta:

### **7.9.1. API's gráficas y Threads**

La mayoría de las API's disponibles para hacer interfaces gráficas desktop fueron diseñadas en un modo no thread-safe [22, 23]. Es decir, que no se puede garantizar un estado consistente si los objetos que pertenecen a la API son accedidos desde múltiples hilos simultáneamente. Por esta razón, todas las actualizaciones a las ventanas, botones, widgets en general son realizados por un único thread denominado GUI Thread (o dispatching thread). Esta decisión de diseño permite que un desarrollador no deba preocuparse en la complejidad de manejar algoritmos y estructuras de sincronización (locks, deadlocks, etc.) que luego son difíciles de testear.

La solución desarrollada en el presente trabajo hace uso de diversos actores (que generalmente corren en un pool de threads) para poder responder asincrónicamente a mensajes que reciben desde el kernel. Algunos de estos mensajes requieren que se produzcan cambios en widgets, por ejemplo al recibir un mensaje que agrega texto al sector de edición. Estos cambios deben realizarse en el GUI Thread y no en el Thread que recibe la operación.

Generalmente, las API's gráficas poseen mecanismos para postergar la ejecución de código para que se ejecute en el Thread de la GUI [24, 25] y evitar los problemas de sincronización.

En la figura 12 se observa el problema descripto y en la figura 13 la solución.

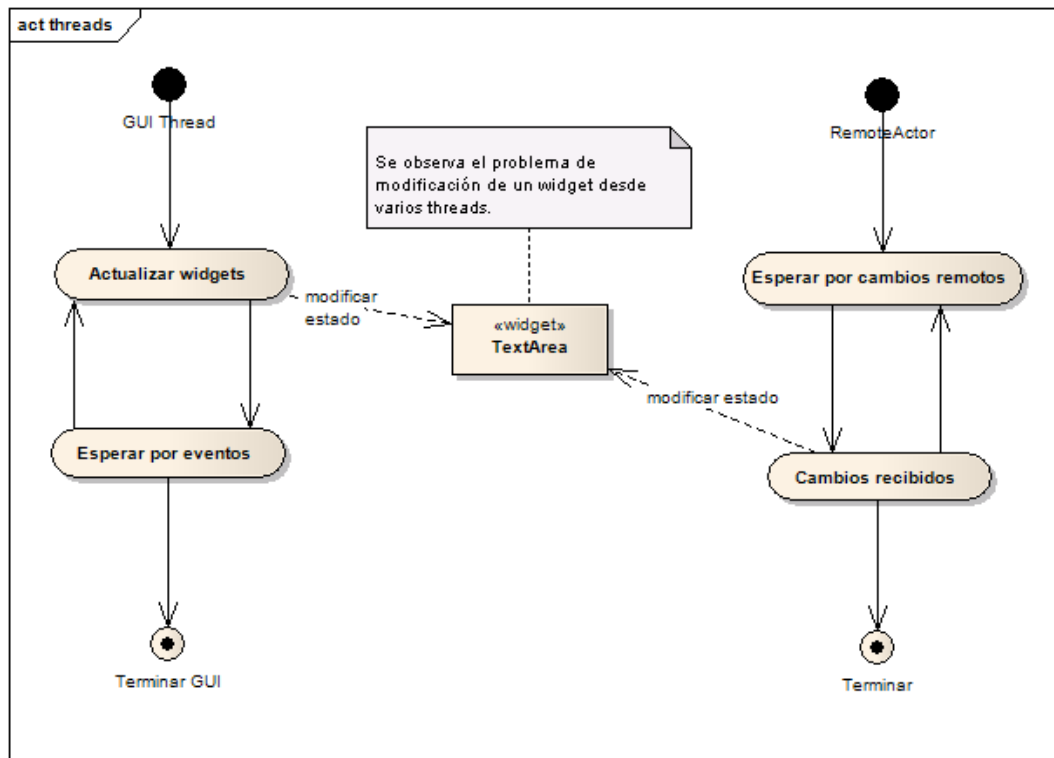


Figura 12: Problemas de múltiples threads junto con el thread de la GUI

### 7.9.2. Editor de texto

Si se desea incorporar la solución en un nuevo editor de texto (por ejemplo un IDE) es necesario tener presente las siguientes consideraciones:

- Se deberá implementar la interfaz `DocumentData` que ofrece un mecanismo para agregar y modificar texto en el editor. A su vez ofrece un mecanismo para mantener la posición del cursor consistente y que no sea movida por inserciones o borrado de texto por otros participantes. Si el componente no soporta selección de texto o no existe un cursor, es posible ignorar las llamadas a estos métodos. La interfaz `DocumentData` se encuentra en el archivo `DocumentData.scala`.
- Se deberá implementar un listener que atrape los eventos del widget de edición de texto y los propague al kernel. La mayoría de las API's gráficas proveen un mecanismo similar.

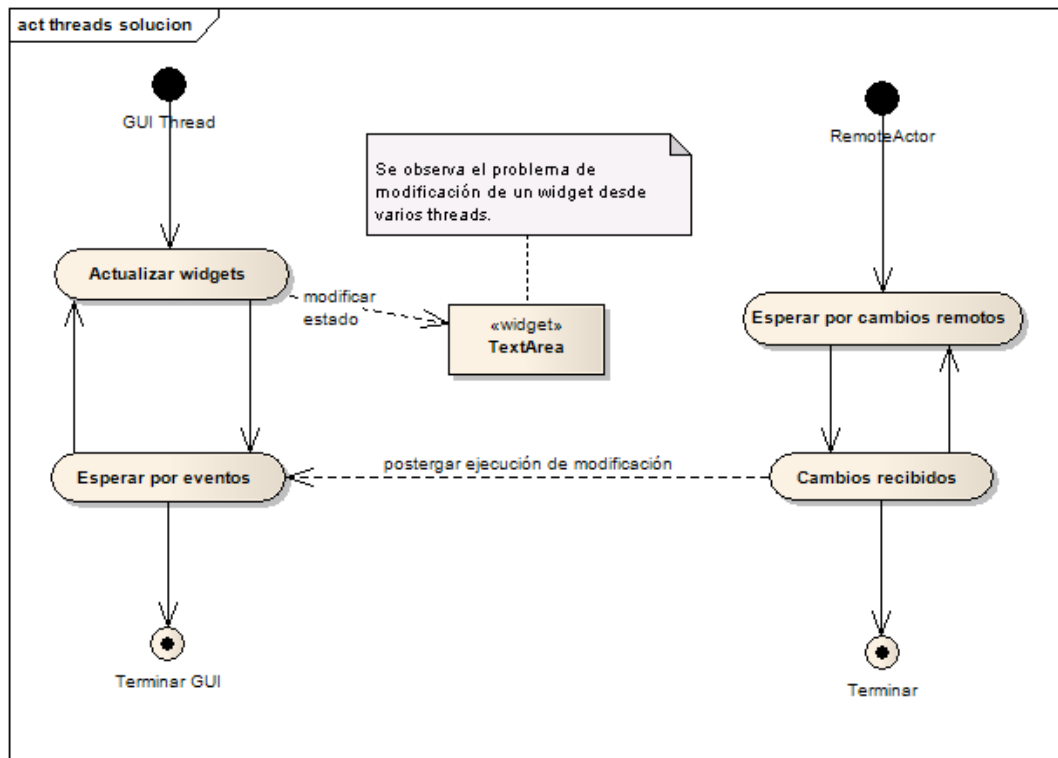


Figura 13: Posible solución de múltiples threads junto con el thread de la GUI

### 7.9.3. Implementación de la interfaz Documents

La implementación de esta interfaz se debe encargar de recibir el mensaje proveniente del kernel y si necesita realizar cálculo intensivo, debe realizarse en otro thread. La causa de esto es que el código se encontrará corriendo en el thread que está corriendo el actor, por lo tanto, los actores deben liberar su thread para que otros actores puedan tomar su tiempo de CPU (propiedades fairness y liveness).



## 8. Riesgos

Los riesgos que se identificaron al inicio del proyecto se detallan a continuación:

### 8.1. ID 1: Demoras en el desarrollo de las funcionalidades por la poca experiencia con Scala

Este riesgo no se materializó, en gran parte ya que se hizo una investigación y capacitación sobre el lenguaje de programación previamente al inicio del desarrollo. Los resultados obtenidos demuestran que fue suficiente para afrontar el proyecto. La comunidad de programadores está en constante crecimiento y es muy activa y ayudó durante el desarrollo con los problemas que surgieron.

### 8.2. ID 2: Demoras en el desarrollo de la integración con Eclipse debido al desconocimiento de la arquitectura del mismo

Se encontró que la documentación disponible de la arquitectura de Eclipse es abundante y de buena calidad por lo cual no hubo mayores inconvenientes en la etapa de desarrollo del *plugin*. Este riesgo no se materializó.

### 8.3. ID 3: Demoras en el proyecto por la no aprobación de la propuesta de trabajo profesional

La aprobación de la propuesta se produjo según lo planeado en el calendario, aproximadamente un mes después del inicio del proyecto. No fue necesario cambiar el alcance del proyecto.

### 8.4. ID 4: Demoras en la definición de la arquitectura produce retrabajo

Se trabajó intensamente en una definición temprana de la arquitectura que resultó estable para el proyecto. Las decisiones arquitecturales fueron tomadas en conjunto con todo el equipo de desarrollo y esto produjo un buen consenso y entendimiento de la misma.

## 8.5. Riesgos no relevados

Durante el desarrollo se materializó un riesgo no contemplado originalmente. La estimación de algunas tareas dentro de los sprints [29] realizados estuvo muy desviada del valor real de horas hombre invertidas. Principalmente el desvío se dio en las tareas de implementación del algoritmo de sincronización. Para éste se estimó un valor mucho menor al real (ver *Sprint Backlog 2*). El desvío fue compensado por la materialización de otro riesgo que fue la sobre estimación de tareas dejando un margen para poder cerrar el *sprint* con toda la funcionalidad convenida.

Los tres sprint realizados fueron cerrados con las tareas de mayor prioridad completas. En algunos casos se pasaron tareas hacia siguientes *sprint*, pero éstas eran de baja prioridad.

## 9. Problemas enfrentados durante el desarrollo

- El lenguaje de programación **Scala** es una tecnología relativamente nueva (nacida en 2003) que si bien es estable cuenta con herramientas poco maduras para su desarrollo. Principalmente, los IDEs disponibles no son tan completos funcionalmente como lo son al desarrollar en el lenguaje Java.
- El desarrollo se comenzó utilizando el IDE **Eclipse** pero la funcionalidad provista era muy limitada por lo que se cambió al **IntelliJ IDEA**. Este resultó ser uno de los IDEs con mayor soporte para el desarrollo en Scala.
- *Scheduling* de actores: para procesos que tienen que ver con I/O se utilizaron *threads* para no bloquear los actores restantes. Se encontró una biblioteca llamada *Akka Actors* [26] que provee una mejor funcionalidad de actores, pero se decidió no cambiar a la misma ya que el proyecto estaba en una etapa de desarrollo avanzada, evitando demoras en el calendario. Esta biblioteca provee facilidades para la configuración del scheduling de los actores.
- Se encontró una herramienta denominada SBT [27] (*Simple Build Tool*) que agiliza la compilación de proyectos desarrollados en Scala, pero la transición de Maven a SBT no resultaba trivial por lo que se descartó. El beneficio de usar esta herramienta es que permite incrementar la velocidad de compilación con respecto a usar el IDE o Maven.

## 10. Metodología de desarrollo

Scrum fue utilizado como metodología de desarrollo para llevar adelante el proyecto. La implementación de esta metodología resultó efectiva. Se alcanzaron los objetivos establecidos en la planificación en un tiempo menor del que se estimó en el calendario propuesto. Esto resultó así ya que durante el desarrollo del tercer sprint, el esfuerzo dedicado al proyecto por los recursos disponibles fue mayor al que originalmente se había planificado, permitiendo completar las tareas pertenecientes al cuarto sprint durante el tercero.

El proyecto se concluyó con tres sprints en los cuales se cumplieron los objetivos detallados en el cuadro 2.

### 10.1. Sprint 1

Durante el primer sprint del desarrollo del presente trabajo, se puso hincapié en definir la arquitectura del mismo para que pueda ser extendida en sprints posteriores.

En la figura 14 se observan las tareas planificadas para el mismo junto con el esfuerzo empleado para cada una de ellas.

ID	Descripción	Esfuerzo Estimado	Esfuerzo Empleado	Status	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	1	2	3	4
1	Modulo servidor: levanta kernel configurado y lo deja disponible	6.5	0	N																						
2	Kernel: Arquitectura base	12	9	F	6	3																				
3	Kernel: modelo de documento	8	4.5	I														3.5								
4	Kernel+cliente: operacion insertar y borrar texto	11	1	F	1																					
5	Kernel: exponer servicios en la red	16	7	F										1				1		2			3			
6	Kernel: Modelado de actores cliente y actores documento.	32	25	F			3	3	4			8	2	1				2		2						
7	Cliente: interfaz de conexion a nucleo	12	9	F										2					2	1	4					2
8	Cliente: interfaz de envio y recepcion de mensajes	16	8	F									2	2					2	1		1				2
9	Gui: diseño básico (ventana de conexion, desconexión, visualización de doc)	12	12	I													4	1	2		2	2	1			
10	Gui: implementación modelo cliente-servidor	13	13	F									3	2					2		2	4			2	2
Total Esfuerzo		138.5	88.5																							
Nota: la estimación esta expresada en horas no productivas.																										
Status: I (iniciado) F (finalizado) N (no comenzado)																										

Figura 14: Tareas planificadas para el sprint n° 1

A continuación listaremos algunas observaciones sobre el desarrollo de este sprint:

- El objetivo del sprint fue alcanzado ya que se obtuvo una buena definición de la arquitectura y el prototipo obtenido cumple con los requisitos detallados en la planificación.
- La duración del sprint fue extendida en dos días ya que al día de cierre existían bugs críticos que no permitían cerrar el mismo. Durante esos días extra se solucionaron esos bugs y se cerró el sprint.

Sprint N°	Objetivos	Fecha de Inicio	Fecha de Fin
1	Obtener un prototipo del producto con funcionalidades básicas. Definir la arquitectura del mismo con el objetivo de ser extendida en sprints posteriores. Los módulos afectados en este sprint son Núcleo, GUI y Cliente.	13/09/2010	01/10/2010
2	Extender el prototipo obtenido del sprint anterior agregándole funcionalidad de la resolución de conflictos generados a partir de la edición concurrente para lograr la convergencia en las partes que participan de la edición. Agregar funcionalidades extras a la UI para la edición de múltiples documentos simultáneamente.	05/10/2010	29/10/2010
3	Obtener un prototipo del plugin para el IDE Eclipse adquiriendo los conocimientos necesarios para la integración de los módulos en el mismo. Como objetivos secundarios se encuentran: investigación de la arquitectura de plugins para Eclipse, mejorar la documentación y refinar la aplicación GUI.	03/11/2010	24/11/2010

Cuadro 2: Objetivos por Sprint

- El esfuerzo real empleado al desarrollo está muy por debajo del planificado. Se había planificado 138.5hs y se emplearon 88.5hs. Este punto fue tenido en cuenta para los posteriores sprints.
- Las tareas que no se cerraron en este sprint fueron planificadas para el siguiente.
- La documentación y tests unitarios al finalizar el sprint no era la esperada, por lo que se planifican horas para estas tareas para el siguiente sprint.

## 10.2. Sprint 2

El segundo sprint tenía por objetivo realizar una investigación sobre los algoritmo de sincronización y la implementación de alguno de ellos en el presente trabajo. El esfuerzo empleado en este sprint fue mayor al sprint anterior y está en el orden del planificado.

Se habían planificado 123hs y el esfuerzo empleado fue de 127.5hs.

En la figura 15 se observan las tareas planificadas para este sprint. Durante la planificación del sprint se decidió agregar una columna que indique la prioridad de la tarea para poder enfocar el equipo de desarrollo en las tareas más prioritarias primero.

ID	Descripción	Prioridad	Esfuerzo Estimado	Esfuerzo Empleado	Status	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	Algoritmo de resolución de conflictos.	1	25	79	F	2	11	6	3	2			4	11	4	6			11	3	11	5						
2	Listar documentos disponibles en la GUI	1	9	4.5	F	1.5			2				1															
3	Suscribirse a un documento del listado de disponibles	1	6	4.5	F				3.5																			
4	GUI: Desconectarse del kernel (refactorizar msgs)	1	6	6.5	F	1.5							1						2	2							1	
5	Recibir contenido inicial del documento al suscribirse a una sesión	1	3	1.5	F	1.5																						
6	Validación de título y usuario único	2	12	3	F														2							1		
7	Crear nuevo documento vacío para compartir	2	15	2.5	F					0.5			2															
8	Crear nuevo documento a partir de archivo para compartir	2	4	1	F									1														
9	GUI: Dejar de escuchar un documento	2	4	1	F								1															
10	Crear módulo cliente	2	2	1	F																							
11	Extraer código del cliente del proyecto GUI	2	12	13	F																							
12	Documentación	2	15	0	N																							
13	Borrar documento. Eliminarlo del kernel o servidor	3	6	6	I																							
14	Crear proyecto servidor	3	2	1	F																							
15	Arrancar servidor usando configuración de archivo	3	2	3	F																							
Total Esfuerzo			123	127.5																								
Acumulado																												
Nota: la estimación esta expresada en horas no productivas.																												
Status: I (Iniciado) F (finalizado) N (no comenzado)																												

Figura 15: Tareas planificadas para el sprint n° 2

En las figuras 16 y 17 se observa el esfuerzo diario por día empleado y el estado de las tareas al finalizar el sprint, respectivamente.

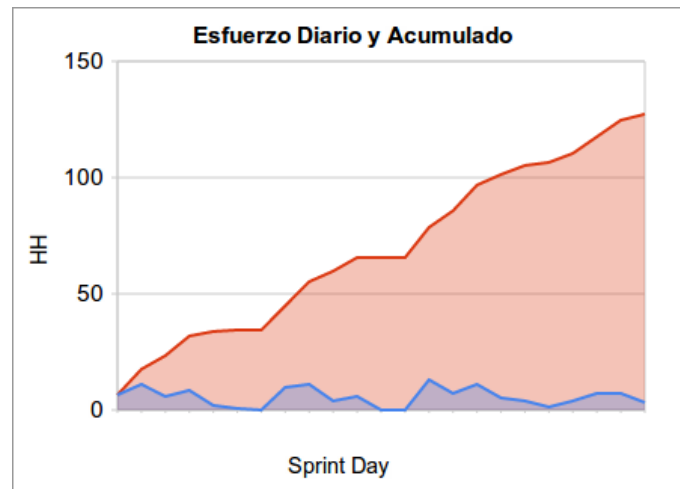


Figura 16: Esfuerzo diario para el sprint n° 2



Figura 17: Estado de las tareas al finalizar el sprint n° 2

### 10.3. Sprint 3

El objetivo del sprint tres fue integrar la solución en algún entorno de desarrollo integrado. Se decidió por el Eclipse IDE ya que es el que más funcionalidades posee en el mundo del desarrollo de Java.

En la figura 18 se observa las tareas planificadas para el sprint tres. En este caso, el esfuerzo empleado fue mayor al planificado ya que se movieron tareas pendientes para el cuarto sprint al tercero para poder cerrar la funcionalidad

sin la necesidad de un cuarto sprint.

ID	Descripción	Prioridad	Esfuerzo Estimado	Esfuerzo Empleado	Status	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	Investigación Plugin	1	25	24	F	2	2					3		1				5	1	6	4						
2	Desarrollo Plugin	1	25	90	F								8	8	3	4	4	6	10	8	7	6	8		4	4	10
3	Conexión y Desconexión (GUI)	1	8	4	F																						4
4	Mensajería (Chat)	2	8	8	F																	6	2				
5	Listado de Usuarios conectados a kernel y docs	2	16	5.5	F [1]		3	1			1.5																
6	Notificación de participantes cuando se unen	2	6	5.5	F [2]						3	2.5															
7	Documentación	2	15	0	N																						
9	Guardar documento en edición (GUI)	3	4	4	F				3		1																
11	Borrar documento (GUI)	2	4	8	F		8																				
12	Retoques Gui	2	4	2	F						2																
Total Esfuerzo			115	151																							
Acumulado																											
Nota: la estimación esta expresada en horas no productivas.																											
Status: I (iniciado) F (finalizado) N (no comenzado)																											

Figura 18: Tareas planificadas para el sprint n° 3

A su vez, se muestran en las figuras 19 y 20 el esfuerzo diario y el estado de las tareas al finalizar el sprint, respectivamente.

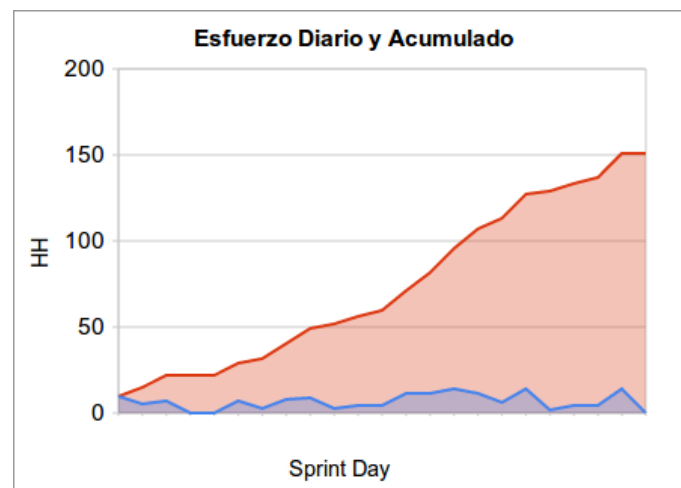


Figura 19: Esfuerzo diario para el sprint n° 3

## 10.4. Otras tareas

Deben tenerse en cuenta en el esfuerzo total realizado unas 85 HH dedicadas a la documentación de la solución y confección del presente documento. Estas tareas quedaron fuera del sprint n° 3 de manera de poder cerrar la fecha del mismo como se había planeado originalmente.

El esfuerzo total dedicado al proyecto fue de 452 HH mientras que el estimado fue de 650 HH.





Figura 20: Estado de las tareas al finalizar el sprint n° 3

De aquí puede verse entonces que el esfuerzo total dedicado al proyecto estuvo por debajo del que originalmente se había estimado y que fue detallado en el documento de Propuesta de Trabajo Profesional.

## 11. Futuras líneas de investigación

En esta sección analizaremos las futuras líneas de investigación y posibles usos de la presente solución:

- Mecanismo para deshacer cambios (undo) [10, 11]: cuando un participante de una sesión de desarrollo se equivoca al teclear, es común que utilice el atajo CTRL-Z para deshacer su último cambio. Debe notarse que la intención del usuario es deshacer su último cambio y no otra posible acción realizada por otro usuario sobre el documento. Actualmente, la solución implementada no respeta esta intención del usuario sino que deshace el último cambio aplicado al documento independientemente de quién lo haya hecho. Esta mejora agrega valor al usuario y es una de las más prioritarias para próximas versiones.

Un mecanismo de undo a implementar debería, dada una operación local ya aplicada, transformar la inversa de ésta en una nueva teniendo en cuenta los efectos que produjeron operaciones subsiguientes a la que se quiere deshacer de forma tal que se logre el efecto deseado por el usuario al aplicarla. Esto significa que al aplicarse se eliminan los efectos de la operación que se quiso deshacer pero preservando las operaciones remotas que fueron aplicadas luego de la misma. La operación de undo será correcta si el estado alcanzado luego de aplicarla es el que se obtendría si nunca se hubiera aplicado la operación a deshacer pero sí las que se aplicaron luego.

- Esquema de autorización: es posible incorporar a la solución mecanismos para restringir o permitir el acceso de determinados usuarios a los sistema de colaboración. En muchas organizaciones existe la necesidad de auditar el trabajo de los empleados por lo cual podría ser útil un registro de usuarios, que actividades realizaron, etc.
- Esquema de autenticación: incorporar a la solución control de acceso de usuarios permitiendo que sólo algunos de ellos accedan a determinados documentos.
- Optimización del uso de red: la reducción de la carga de red mejoraría el tiempo de respuesta de los demás participantes generando una sensación de mayor interactividad.
- Integración con otros IDEs: el plugin desarrollado en este trabajo fue implementado para el IDE Eclipse. Éste es actualmente uno de los más utilizado por desarrolladores en todo el mundo. La arquitectura de los

módulos base fue pensada para ser independiente de la aplicación que los integre. De esta forma es posible desarrollar plugins para otros entornos integrados de desarrollo que permiten la inclusión de los mismos como por ejemplo: IntelliJ IDEA.

- Extensión de la solución para colaboración en tiempo de real sobre otros tipos de documentos: la solución implementada en este trabajo se limita a la colaboración en tiempo real sobre un modelo que representa documentos de texto. Sin embargo la teoría de la transformada operacional (OT) es aplicable a cualquier modelo de objeto en general. De esta manera sería posible llevar la base de la implementación de este trabajo a otros tipos de modelos. Ejemplos de modelos para los que se han implementado aplicaciones colaborativas son áreas de dibujo, documentos CAD y controles de interfaz gráfica entre otros.
- Cifrado de los datos: hoy en día el valor de la información es cada vez más importante dentro de las organizaciones. Por este motivo debe tenerse en cuenta que si se va a utilizar una red pública o que pueda estar expuesta al monitoreo de terceros es sumamente importante no exponer datos sensibles en la misma sin las debidas precauciones que esto requiere. Bajo esta premisa sería deseable que la información transmitida por la aplicación sea enviada por la red utilizando algún esquema de cifrado. Esta modificación se podría implementar añadiendo una capa que encapsule el cifrado y descifrado de los datos, haciéndola intercambiable y transparente a la aplicación.
- Tráfico a través de un proxy: analizar una posible solución para que el producto pueda funcionar detrás de un proxy HTTP. Es muy común que en organizaciones se instale un proxy HTTP para aumentar la performance o para restringir el acceso a determinados sitios. Como consecuencia se prohíbe todo el tráfico que no sea HTTP.

## 12. Conclusiones

Es importante simplificar el modelo que se utiliza para la programación de aplicaciones distribuidas en tiempo real. Se requieren construcciones que separen los conceptos de infraestructura (sincronización, hilos, bloqueos, etc.) del dominio de la aplicación. La abstracción de estos conceptos resulta clave para el desarrollo de sistemas más comprensibles y mantenibles. Esto permite a su vez simplificar el proceso de testing o de aseguramiento de calidad generando productos más confiables y de mayor valor para el cliente.

Aquí es dónde el modelo de concurrencia basado en Actores presenta un gran futuro. La implementación provista en la Scala Library, que es usada en la presente solución, es completamente funcional.

Los lenguajes que hacen uso de la máquina virtual de Java (JVM) están apuntando a simplificar algunas tareas que en el lenguaje de programación Java pueden resultar engorrosas. Entre otras características intentan aplicar el principio DRY (Don't Repeat Yourself, en castellano No repitas) y llevan el lenguaje a un alto nivel integrando conceptos de paradigmas de programación que antes eran meramente de uso académico incorporándolos con éxito en el ambiente corporativo o *mainstream*. Como ejemplo se observa que la utilización de Scala en el desarrollo de este trabajo fue exitosa produciendo un código fuente conciso y de alto nivel.

La utilización de modelos para la implementación de sistemas colaborativos de tiempo real está en plena evolución. Existen gran cantidad de publicaciones en las cuales se presentan nuevas alternativas, agregados y mejoras a los primeros enfoques desarrollados. Esto presenta un panorama alentador en vista de los resultados que puede tener la aplicación práctica de estas tecnologías en el futuro cercano.

Cada vez pueden verse más aplicaciones que experimentan con la incorporación de características de colaboración en tiempo real de forma nativa.

Estos avances y tendencias están logrando su objetivo de permitir la interacción de un grupo de trabajo de manera coordinada y sencilla permitiendo acelerar, enriquecer y hacer más productivo su trabajo.

## 13. Fuentes

En esta sección se presenta el código fuente de las principales clases correspondientes al trabajo profesional que son referidas a lo largo de este texto.

El código fuente de este proyecto se encuentra liberado bajo la licencia GNU GPL v3 [30] y puede ser encontrado en el repositorio de GitHub [21]. Esta licencia implica que el desarrollo es software libre.

La misma permite que el código fuente de este trabajo profesional sea usado en otros proyectos siempre y cuando éstos utilicen la misma licencia, es decir que también sean libres. Por el contrario, no permite que desarrollos cerrados hagan uso de este proyecto sin autorización de los autores originales.

### 13.1. RemoteMessages.scala

```
package ar.noxit.paralleleditor.common.messages

import scala.Serializable

/**
 * Clase base de los mensajes remotos
 */
@Serializable
sealed trait BaseRemoteMessage

/**
 * Mensajes que van hacia el kernel y que deben convertirse antes de ser enviados
 */
@Serializable
sealed trait ToKernel

/**
 * Mensajes enviados desde el cliente que tienen destino al kernel, son todos request
 */
@Serializable
sealed trait Request

/**
 * Mensaje proveniente del kernel que son respuestas a mensajes
 */
@Serializable
sealed trait Response

/**
 * Nivel documento
 */

/**
 * Clase base para las operaciones sobre documentos
 */
sealed trait RemoteOperation extends BaseRemoteMessage

/**
 * Agregar texto
 */
case class RemoteAddText(val text: String, val startPos: Int, val pword: List[Int]) extends RemoteOperation

/**
 * Borrar texto
 */
case class RemoteDeleteText(val startPos: Int, val size: Int) extends RemoteOperation

/**
 * Null operation
 */
case class RemoteNullOpText() extends RemoteOperation

/**
 * A nivel Sincronismo
 */
```

```

/**
 * información de sincronización del documento
 */
@serializable
case class SyncStatus(val myMsgs: Int, val otherMessages: Int)

/**
 * Sincronizar operaciones
 */
case class SyncOperation(val syncSatus: SyncStatus, val payload: RemoteOperation) extends BaseRemoteMessage

/**
 * A nivel kernel
 */

/**
 * Operacion sobre un determinado documento
 */
case class RemoteDocumentOperation(val docTitle: String, val payload: SyncOperation) extends BaseRemoteMessage

/**
 * Pide un nuevo documento
 */
case class RemoteNewDocumentRequest(val title: String, val initialContent: String = "") extends BaseRemoteMessage with ToKernel with Request

/**
 * Suscribirse a un doc existe
 */
case class RemoteSubscribeRequest(val title: String) extends BaseRemoteMessage with ToKernel with Request

/**
 *
 */
case class RemoteDocumentTitleExists(val offenderTitle: String) extends BaseRemoteMessage with Response

/**
 * Suscripcion aceptada
 */
case class RemoteDocumentSubscriptionResponse(val docTitle: String, val initialContent: String) extends BaseRemoteMessage with Response

/**
 * Ya existe subscripcion
 */
case class RemoteDocumentSubscriptionAlreadyExists(val offenderTitle: String) extends BaseRemoteMessage with Response

/**
 * No existe subscripcion
 */
case class RemoteDocumentSubscriptionNotExists(val offenderTitle: String) extends BaseRemoteMessage with Response

/**
 * No se puede borrar el documento, está en uso
 */
case class RemoteDocumentInUse(val docTitle: String) extends BaseRemoteMessage with Response

/**
 * El doc se borró ok
 */
case class RemoteDocumentDeletedOk(val docTitle: String) extends BaseRemoteMessage with Response

/**
 * El titulo no existe
 */
case class RemoteDocumentDeletionTitleNotExists(val docTitle: String) extends BaseRemoteMessage with Response

/**
 * Pide listado de usuarios
 */
case class RemoteUserListRequest() extends BaseRemoteMessage with Request with ToKernel

/**
 * Pide listado de documentos
 */
case class RemoteDocumentListRequest() extends BaseRemoteMessage with ToKernel with Request

/**
 * Respuesta listado de usuarios
 */
case class RemoteUserListResponse(val usernames: Map[String, List[String]]) extends BaseRemoteMessage with Response

/**
 * Respuesta de listado de documentos

```

```

    */
case class RemoteDocumentListResponse(val docList: List[String]) extends BaseRemoteMessage with Response

/**
 * Pide desuscribirse a un doc
 */
case class RemoteUnsubscribeRequest(val title: String) extends BaseRemoteMessage with Request

/**
 * Pide al kernel que borre un documento
 */
case class RemoteDeleteDocumentRequest(val docTitle: String) extends BaseRemoteMessage with Request with ToKernel

/**
 * Pide login
 */
case class RemoteLoginRequest(val username: String) extends BaseRemoteMessage with Request

/**
 * Rta de login OK
 */
case class RemoteLoginOkResponse() extends BaseRemoteMessage with Response

/**
 * Rta de Login Erroneo
 */
trait RemoteLoginRefusedRemoteResponse extends BaseRemoteMessage with Response

/**
 * Nombre de usuario tomado
 */
case class UsernameAlreadyExistsRemoteResponse() extends RemoteLoginRefusedRemoteResponse with Response

/**
 * Pedido de logout
 */
case class RemoteLogoutRequest() extends BaseRemoteMessage

/**
 * Nueva sesión iniciada
 */
case class RemoteNewUserLoggedIn(val username: String) extends BaseRemoteMessage with Response

/**
 * Sesión cerrada
 */
case class RemoteUserLoggedOut(val username: String) extends BaseRemoteMessage with Response

/**
 * Nueva suscripcion a un documento
 */
case class RemoteNewSubscriberToDocument(val username: String, val docTitle: String) extends BaseRemoteMessage with Response

/**
 * Desuscripcion a un documento
 */
case class RemoteSubscriberLeftDocument(val username: String, val docTitle: String) extends BaseRemoteMessage with Response

/**
 * desuscripcion del document ok
 */
case class RemoteSubscriptionCancelled(val docTitle: String) extends BaseRemoteMessage with Response

/**
 * Cuando no existe documento en la suscripcion
 */
case class RemoteDocumentNotExists(val offenderTitle: String) extends BaseRemoteMessage with Response

/**
 * Mensaje de chat de otro usuario
 */
case class RemoteChatMessage(val username: String, val message: String) extends BaseRemoteMessage with Response

/**
 * Send message
 */
case class RemoteSendChatMessage(val message: String) extends BaseRemoteMessage with Request with ToKernel

```

## 13.2. ClientMessages.scala

```
package ar.noxit.paralleleditor.client

import actors.Actor

/**
 * Mensajes que se envian entre un Cliente y el kernel
 */

/**
 * Le pide al kernel que lo desloguee
 */
case class Logout()

/**
 * Mensajes entre elg ClientActor y el RemoteServerProxy
 */
case class FromKernel(val msg: Any)
case class ToKernel(val msg: Any)
case class RegisterRemoteActor(val remote: Actor)
```

## 13.3. DocumentMessages.scala

```
package ar.noxit.paralleleditor.kernel.messages

import ar.noxit.paralleleditor.kernel.Session
import ar.noxit.paralleleditor.common.operation.EditOperation
import ar.noxit.paralleleditor.common.Message

/**
 * Mensajes que se envian entre el actor del documento y el kernel.
 */

case class TerminateDocument()

case class ProcessOperation(val who: Session, val m: Message[EditOperation])

case class SubscriberCount()
case class Subscribe(val who: Session)
case class Unsubscribe(val who: Session)
case class SilentUnsubscribe(val session: Session)
case class Close(val session: Session)
case class DocumentDeleted(val docTitle: String)

case class DocumentUserListRequest()

case class DocumentUserListResponse(val docTitle: String, val users: List[String])

/**
 * La envia el doc actor para que el client actor la retransmita al cliente remoto
 */
case class PublishOperation(val docTitle: String, val m: Message[EditOperation])
```

## 13.4. SocketNetworkConnection.scala

```
package ar.noxit.paralleleditor.common.network

import java.net.Socket
import java.io.{ObjectOutputStream, OutputStream, ObjectInputStream, InputStream}
import ar.noxit.paralleleditor.common.logger.Loggable

class SocketNetworkConnection(private val socket: Socket) extends NetworkConnection with Loggable {
  trace("Socket network connection created")

  override def messageOutput = new SocketMessageOutput(socket.getOutputStream)

  override def messageInput = new SocketMessageInput(socket.getInputStream)

  override def close = {
    trace("Network connection closed")
    socket.close
  }
}

class SocketMessageInput(private val inputStream: InputStream) extends MessageInput {
```



```

    private val objectInput = new ObjectInputStream(inputStream)

    override def readMessage = objectInput readObject
}

class SocketMessageOutput(private val outputStream: OutputStream) extends MessageOutput {
    private val objectOutputStream = new ObjectOutputStream(outputStream)

    override def writeMessage(message: Any) = objectOutputStream writeObject message
}

```

## 13.5. JupiterSynchronizer.scala

```

package ar.noxit.paralleleditor.common

import operation._

class BasicXFormStrategy extends XFormStrategy {
    override def xform(ops: (EditOperation, EditOperation)) = {
        if (ops == null)
            throw new IllegalArgumentException("ops cannot be null")

        ops match {
            case (c: AddTextOperation, s: AddTextOperation) =>
                xform(c, s)
            case (c: DeleteTextOperation, s: DeleteTextOperation) =>
                xform(c, s)
            case (c: AddTextOperation, s: DeleteTextOperation) =>
                xform(c, s)
            case (c: DeleteTextOperation, s: AddTextOperation) =>
                xform(s, c).swap
            case (c: EditOperation, s: EditOperation) if c.isInstanceOf[NullOperation] || s.isInstanceOf[NullOperation] =>
                (c, s)
        }
    }

    /**
     * Caso agregar-agregar
     */
    protected def xform(c: AddTextOperation, s: AddTextOperation): (EditOperation, EditOperation) =
        (simpleXForm(c, s), simpleXForm(s, c))

    /**
     * Implementación según paper
     * Achieving Convergence with Operational
     * Transformation in Distributed Groupware Systems
     */
    protected def simpleXForm(c: AddTextOperation, s: AddTextOperation) = {
        if (c.text.size != 1 || s.text.size != 1) throw new UnsupportedOperationException("add size must be 1")

        val p1 = c.startPos
        val p2 = s.startPos
        val c1 = c.text
        val c2 = s.text
        val w1 = c.pword

        val alfa1 = pw(c)
        val alfa2 = pw(s)

        if (menor(alfa1, alfa2) || (igual(alfa1, alfa2) && c1 < c2))
            c
        else if (mayor(alfa1, alfa2) || (igual(alfa1, alfa2) && c1 > c2))
            new AddTextOperation(c1, p1 + c2.length, p1 :: w1)
        else
            c
    }

    /**
     * Caso borrar-borrar
     * para operaciones de borrado de 1 caracter
     */
    protected def xform(c: DeleteTextOperation, s: DeleteTextOperation): (EditOperation, EditOperation) = {
        if (s.size != 1 || c.size != 1) throw new UnsupportedOperationException("Delete size must be 1")

        val p1 = c.startPos
        val p2 = s.startPos

        if (p1 < p2)
            (c, new DeleteTextOperation(p2 - 1, s.size))
        else if (p1 > p2)

```

```

        (new DeleteTextOperation(p1 - 1, c.size), s)
    else
        (new NullOperation, new NullOperation)
}

/**
 * Caso agregar-borrar
 * la implementación contempla solo operaciones de borrado
 * de un caracter
 */
protected def xform(c: AddTextOperation, s: DeleteTextOperation): (EditOperation, EditOperation) = {
    if (c.text.size != 1) throw new UnsupportedOperationException("add size must be 1")
    if (s.size != 1) throw new UnsupportedOperationException("Delete size must be 1")

    val p1 = c.startPos
    val p2 = s.startPos
    val pw = c.pword

    if (p1 > p2)
        (new AddTextOperation(c.text, p1 - 1, p1 :: pw), s)
    else if (p1 < p2)
        (c, new DeleteTextOperation(p2 + c.text.length, s.size))
    else
        (new AddTextOperation(c.text, p1, p1 :: pw), new DeleteTextOperation(p2 + c.text.length, s.size))
}

/**
 * público para testing
 */
def pw(op: EditOperation) = {
    op match {
        case at: AddTextOperation => {
            // primer caso si w == vacio, con w = pword
            val p = at.startPos
            val w = at.pword

            if (w.isEmpty)
                List(p)
            else if (!w.isEmpty && (p == current(w) || (p - current(w)).abs == 1))
                p :: w
            else
                List()
        }
        case dt: DeleteTextOperation => {
            val p = dt.startPos
            List(p)
        }
        case o: NullOperation => List()
    }
}

protected def current(pword: List[Int]) = pword.head

private def getRangeFor(o: DeleteTextOperation) = o.startPos to (o.startPos + o.size)

def menor(a: List[Int], b: List[Int]) = comparar(a, b, {(v1, v2) => v1 < v2})

def mayor(a: List[Int], b: List[Int]) = comparar(a, b, {(v1, v2) => v1 > v2})

def igual(a: List[Int], b: List[Int]) = comparar(a, b, {(v1, v2) => v1 == v2})

private def comparar(a: List[Int], b: List[Int], comp: (Int, Int) => Boolean) = {
    val tuples = a.zip(b)
    val result = tuples.dropWhile {t => t._1 == t._2}
    if (result.isEmpty)
        // aca una es mas larga q la otra
        comp(a.size, b.size)
    else {
        val head = result.head
        comp(head._1, head._2)
    }
}
}

```

## 13.6. BasicXFormStrategy.scala

```

package ar.noxit.paralleleditor.common

import logger.Loggable
import operation.EditOperation

```

```

case class Message[Op](val op: Op, val myMsgs: Int, val otherMsgs: Int)

abstract class JupiterSynchronizer[Op] extends Loggable {
  /**
   * n° de mensajes originados localmente
   */
  private var _myMsgs = 0

  /**
   * n° de mensajes que llegaron del afuera
   */
  private var _otherMsgs = 0

  /**
   * lista de mensajes que se generaron y enviaron localmente
   */
  private var outgoingMsgs = Map[Int, Op]()

  /**
   * Getter para testing
   */
  def myMsgs = _myMsgs

  /**
   * Getter para testing
   */
  def otherMsgs = _otherMsgs

  /**
   * La operación ya fue aplicada antes de llamarse a este método
   */
  def generate(op: Op, send: Message[Op] => Unit) {
    trace("Generating message")

    // enviar mensaje a la otra parte
    send(Message(op, _myMsgs, _otherMsgs))

    outgoingMsgs = outgoingMsgs.updated(_myMsgs, op)
    _myMsgs = _myMsgs + 1

    trace("estado actual %d %d", _myMsgs, _otherMsgs)
  }

  def receive(message: Message[Op], apply: Op => Unit) {
    trace("Message received " + message)

    // filtro mensajes anteriores al recibido (acknowledged messages)
    outgoingMsgs = outgoingMsgs filterKeys (_ >= message.otherMsgs)

    val ordenedMsgs = outgoingMsgs.toArray.sortBy {_. _1}

    trace("original op %s", message.op)
    // calculo la transformada de la operacion a realizar
    val finalOp = (message.op /: ordenedMsgs) {
      (transformedOp, currentListElement) => {
        val currentOp = currentListElement._2
        val transformedOps = xform(currentOp, transformedOp)

        outgoingMsgs = outgoingMsgs.updated(currentListElement._1, transformedOps._1)

        transformedOps._2
      }
    }
    trace("fixed op %s", finalOp)
    apply(finalOp)

    _otherMsgs = _otherMsgs + 1
    trace("estado actual %d %d", _myMsgs, _otherMsgs)
  }

  protected def xform(c: Op, s: Op): (Op, Op)
}

class EditOperationJupiterSynchronizer(private val xformStrategy: XFormStrategy) extends JupiterSynchronizer[EditOperation] {
  override protected def xform(c: EditOperation, s: EditOperation) = xformStrategy.xform(c, s)
}

trait SendFunction {
  def send(message: Message[EditOperation])
}

trait ApplyFunction {

```

```

    def apply(operation: EditOperation)
  }

class JEditOperationJupiterSynchronizer(val adaptedSync: JupiterSynchronizer[EditOperation]) {
  def generate(op: EditOperation, fun: SendFunction) =
    adaptedSync.generate(op, {m => fun.send(m)})

  def receive(message: Message[EditOperation], fun: ApplyFunction) =
    adaptedSync.receive(message, {op => fun.apply(op)})
}

```

## 13.7. DocumentData.scala

```

package ar.noxit.paralleleditor.common.operation

trait Caret {
  def offset: Int
  def selectionLength: Int
  def change(offset: Int, selectionLength: Int)
}

trait DocumentData {
  def data: String
  def replace(offset: Int, length: Int, newText: String)
  val caret: Caret
}

```

## 14. Manual de Usuario

El manual que se provee a continuación indica de forma sencilla cómo hacer uso del plugin desarrollado. Es un extracto de la wiki [32] del repositorio del proyecto y se redactó en idioma inglés ya que este sitio es de acceso internacional y se pretende que esté a disposición de un mayor número de usuarios.

### 14.1. Installation

Parallel editor is provided as a plugin for the Eclipse integrated development environment. Installation can be performed using the built-in mechanism within it or by manually downloading the plugin files and placing them in the proper plugin folder.

#### 14.1.1. Installation within Eclipse

1. Select “*Install new software*” under the Eclipse “*Help*” menu.
2. Click the “*Add*” button in order to add the location of Parallel Editor plugin repository.
3. Enter `http://maurociancio.github.com/parallel-editor/update-site/` in the location address and click “*Ok*”. You may optionally enter a description for the repository.



Figura 21: Add Site dialog.

4. You should now see ParallelEditor plugin listed (Fig. 22). Select it and click “*Next*”

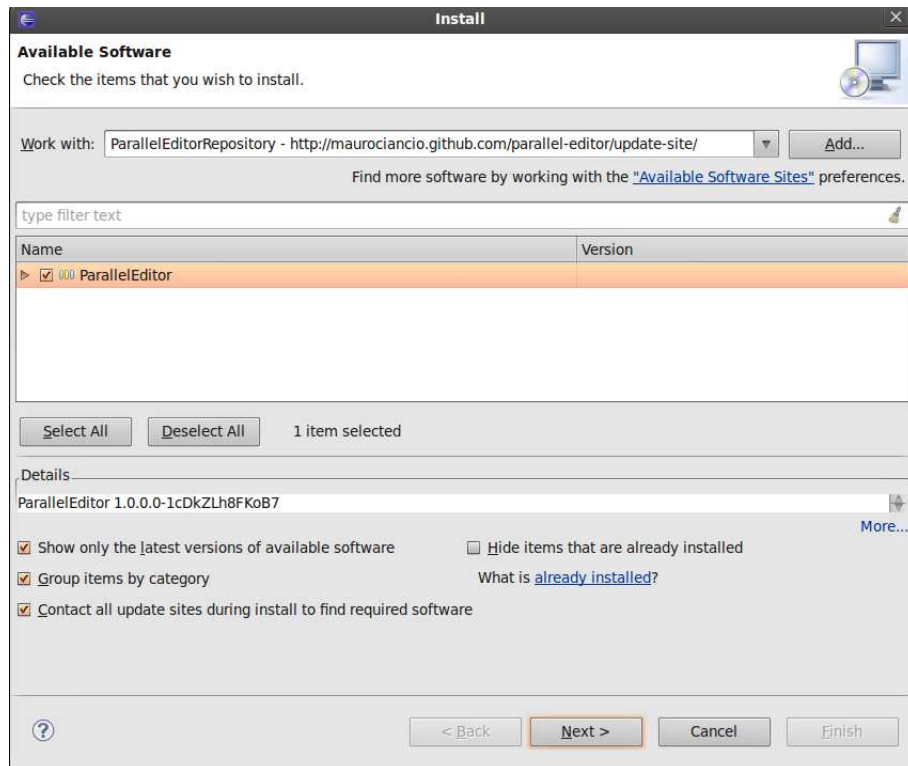


Figura 22: Install new software dialog.

5. After this step the process is straightforward. Confirm your selection, read and accept the license provided if agreed (Fig. 23) and click “*Finish*” to complete the installation.

#### 14.1.2. Manual installation

An optional way to install the plugin is available, however this method is not recommended. To perform a manual installation follow these steps.

1. Clone the project from:  
`git://github.com/maurociancio/parallel-editor.git.`
2. Build the plugin dependencies by issuing in your console: “`mvn clean install`”.
3. Use the script in the base folder in order to copy the dependencies to the Eclipse Plugin. Type: `./copy_libs_to_eclipse_plugin.sh.`

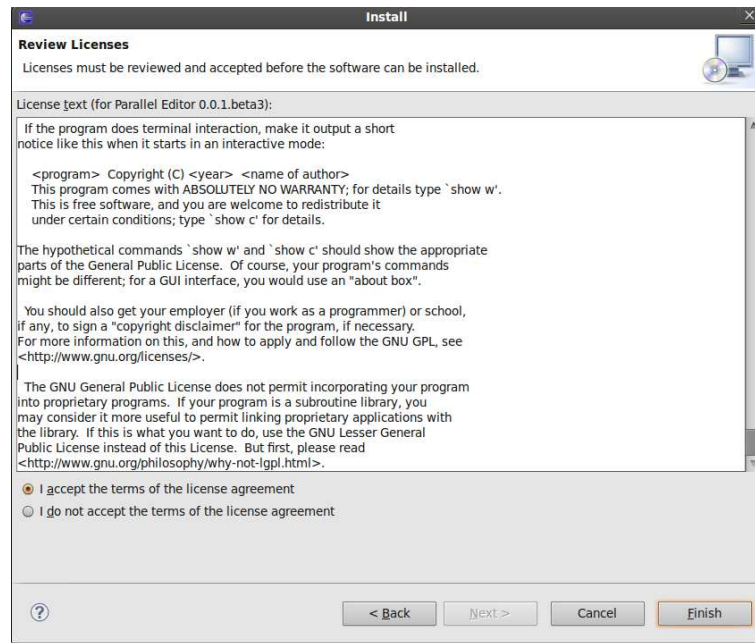


Figura 23: License agreement.

4. Build the Eclipse Plugin .jar using the Update Site project located in the folder: **parallel-editor-eclipse-extras**.
5. Place the .jar file generated in your Eclipse plugin folder (usually /plugins) under the Eclipse installation folder.
6. Restart Eclipse.

ParallelEditor plugin should now be available.

## 14.2. Usage

### 14.2.1. Prerequisites

If you are intended to use it over Internet, you have to check that there is connectivity between you and your peers. Parallel Editor uses a port to transmit data. If you are behind a router please bear in mind that you will have to forward it.

If you are intended to use it over a LAN, you are ready to go. Usually there are no problems in using it over a LAN.

For installation instructions see the Installation section.

### 14.2.2. I have it installed, now what?

Once you have it installed you're ready to go.

Let's suppose you're working on a Java class (or whatever other language) and you need some help. A partner of yours wants to help you with that problem. Both of you are working on the same project, so both have the project checked out from a SCM in their workstations.

So, you are going to initiate a document editing session. Open up your Java file and right click over it. A 'Share thru PE' menu should appear right there.

This is how the menu should be seen:

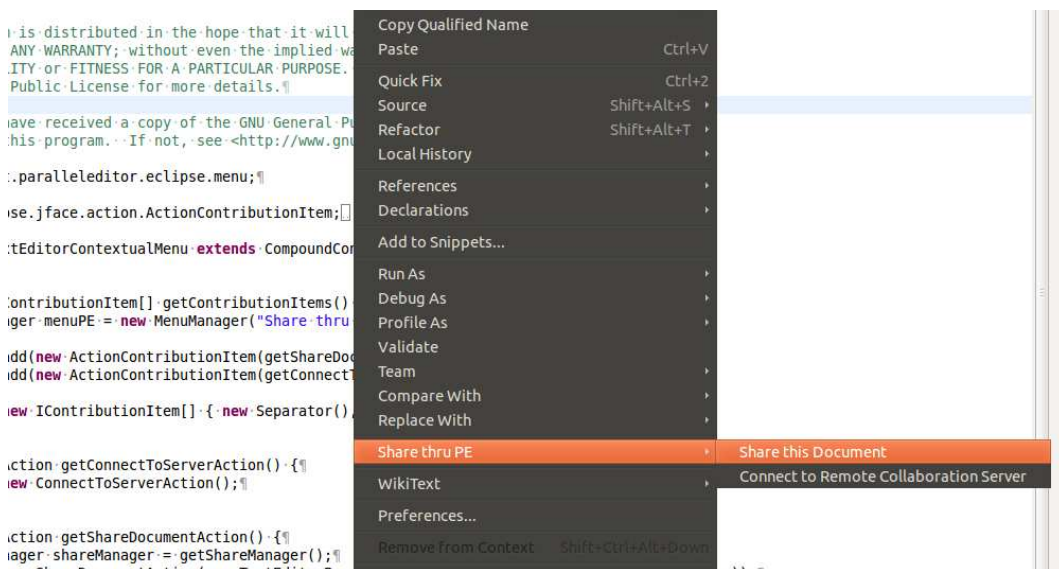


Figura 24: Share menu.

In this menu, you've got two options. The first one 'Share this document' will create a local server and share the current file you're editing. After clicking this menu, the Share view will show up.

This view shows all the information you need about your shares and connections. Here we have the screenshot:

You can see the port currently used. In this case, port 5000 is being used. You can give your ip address and port to your peers and they can join the session. There is no restriction in how many users can edit at the same time.

Also, you can see the users connected to your server and the files being shared.



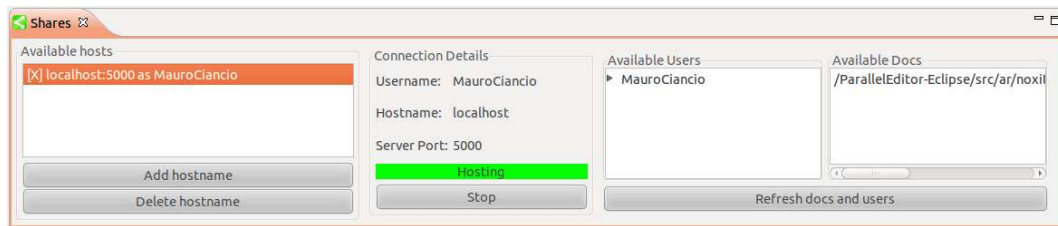


Figura 25: Share views.

### 14.2.3. Connecting to other sessions

If a session is already started, you may want to join it. The only thing you need is the ip address and port. When you found it out, create a new connection to that server by clicking in 'Add hostname' button in the Share view.

When the connection is made, the files shared will appear in the 'available documents' section. By double-clicking it, you're in. Every keystroke you make, will be transmited over the network to the peers and they will see the change instantly. An optimistic scheme is used, so no lock is acquired over the document.

### 14.2.4. Configuration

Some options are availabe in Eclipse Preferences. You can tweak your default port and default username. Here we have the screenshot:

**Port:** indicates the port on which the local server will listen for remote conections. Make sure there is no firewall software blocking connections to it.

**Username:** this is the name that will be used to log on the local server and that your peers will see when connected to it.

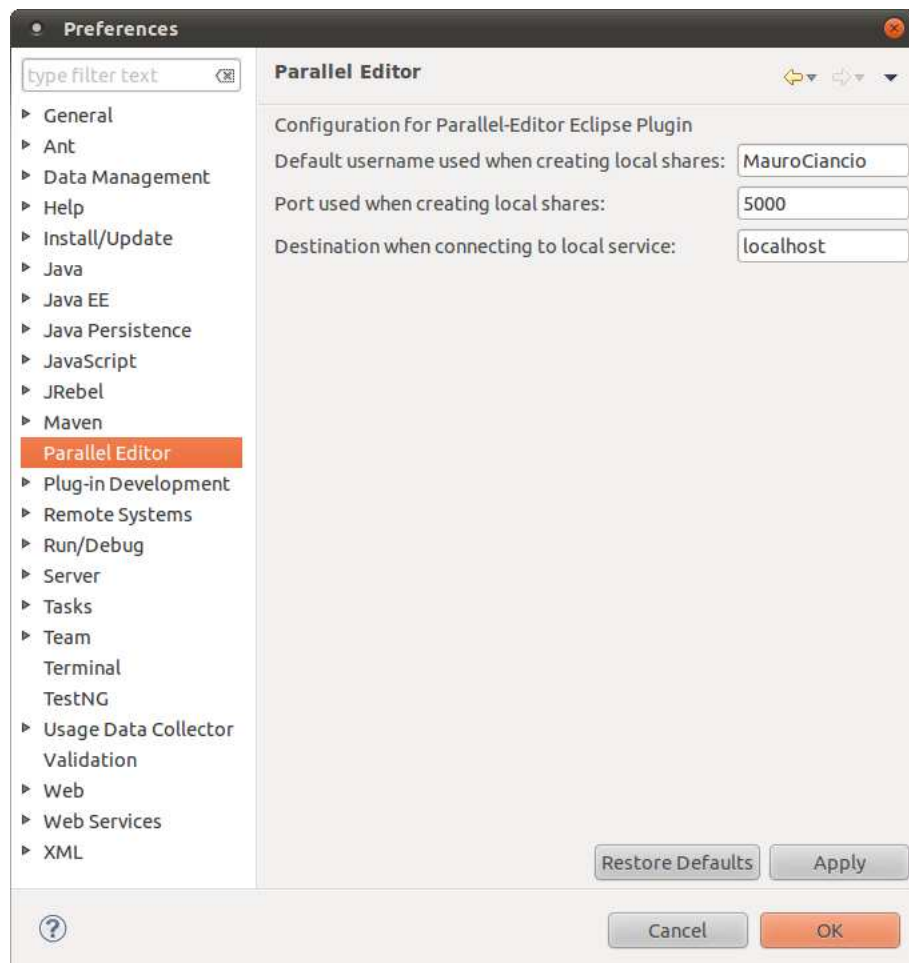


Figura 26: Preferences dialog

## Referencias

- [1] Pair-Programming.  
[http://en.wikipedia.org/wiki/Pair\\_programming](http://en.wikipedia.org/wiki/Pair_programming).
- [2] Eclipse IDE.  
<http://www.eclipse.org/>.
- [3] Ciancio, Gilioli, *Visión Trabajo Profesional*. Facultad de Ingeniería. Universidad de Buenos Aires.
- [4] Ciancio, Gilioli, *Propuesta de Trabajo Profesional - Paralell Editor*. Facultad de Ingeniería. Universidad de Buenos Aires.
- [5] Imine, Molli, Oster, Rusinowitch *Achieving Convergence With Operational Transformation in Distributed Groupware Systems*. Institut National de Recherche en Informatique et en Automatique. Rapport de recherche n° 5188. Mayo 2004.
- [6] Nichols, Curtis, Dixon and Lamping, *High-Latency, Low-Bandwidth Windowing in the Jupiter Collaboration System*. Xerox Palo Alto Research Center.
- [7] Haller and Odersky. *Actors That Unify Threads and Events*. Programming Methods Lab (LAMP). École Polytechnique Fédérale de Lausanne (EPFL). Switzerland.
- [8] Haller and Odersky. *Event-Based Programming without Inversion of Control*. Programming Methods Lab (LAMP). École Polytechnique Fédérale de Lausanne (EPFL). Switzerland.
- [9] Wang, Mash and Lassen *Google Wave Operational Transformation*. Versión: 1.1. Julio 2010.
- [10] C. Sun. *Undo as concurrent inverse in group editors* ACM Transactions on Computer-Human Interaction Vol. 9, No. 4, pp. 309 – 361, Diciembre. 2002.
- [11] D. Sun, C. Sun *Context-based Operational Transformation in Distributed Collaborative Editing Systems* IEEE Transactions on Parallel and Distributed Systems Vol. 20, No. 10, pp. 1454 – 1470, Octubre. 2009.
- [12] *Google Docs*. Google Inc., <http://docs.google.com>.
- [13] *Google Wave*. Google Inc., <http://wave.google.com>.

- [14] Corvalius, *BeWeeVee*. <http://www.beweevee.com>.
- [15] Mustafa K. Isik, *COLA*. [http://wiki.eclipse.org/RT\\_Shared\\_Editing](http://wiki.eclipse.org/RT_Shared_Editing).
- [16] Venkat Subramaniam, *Programming Scala: Tackle Multi-Core Complexity on the Java Virtual Machine*. Pragmatic Bookshelf, ISBN: 193435631X. Julio 2009.
- [17] Ken Schwaber, *Agile Software Development with Scrum*. Prentice Hall, Octubre 2001.
- [18] JUnit, version: 4.8.2. *Resources for Test Driven Development*. <http://junit.org>.
- [19] EasyMock, versión 2.4 *Dynamic Mock Object generator*. <http://sourceforge.net/projects/easymock>.
- [20] TestNG, versión: 5.14. *Cédric Beust* <http://testng.org>.
- [21] Github Social Coding. *Repositorio de código fuente del proyecto*. <https://github.com/maurociancio/parallel-editor>.
- [22] Oracle, *Threads and Swing*. <http://java.sun.com/products/jfc/tsc/articles/threads/threads1.html>.
- [23] Eclipse Helios, *Threading issues*. [http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/swt\\_threading.htm](http://help.eclipse.org/helios/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/swt_threading.htm).
- [24] Swing API, *Invoke Later*. [http://download.oracle.com/javase/1.4.2/docs/api/javax/swing/SwingUtilities.html#invokeLater\(java.lang.Runnable\)](http://download.oracle.com/javase/1.4.2/docs/api/javax/swing/SwingUtilities.html#invokeLater(java.lang.Runnable)).
- [25] SWT API, *Async Exec*. <http://book.javanb.com/swt-the-standard-widget-toolkit/ch05lev1sec7.html>.
- [26] Akka Actors. <http://akka-source.org>.
- [27] Simple Build Tool. <http://code.google.com/p/simple-build-tool>.
- [28] Mock Objects. [http://en.wikipedia.org/wiki/Mock\\_object](http://en.wikipedia.org/wiki/Mock_object).

- [29] Sprint - Scrum.  
*[http://en.wikipedia.org/wiki/Sprint\\_\(scrum\)](http://en.wikipedia.org/wiki/Sprint_(scrum))*.
- [30] GPL - General Public Licence.  
*<http://www.gnu.org/licenses/gpl.html>*.
- [31] Scala - Large amount of Actores.  
*<http://scala-programming-language.1934581.n4.nabble.com/Large-amounts-of-actors-td3019390.html>*.
- [32] Wiki del proyecto.  
*<https://github.com/maurociancio/parallel-editor/wiki>*.



# Editor Paralelo

(Parallel Editor)



Trabajo Profesional

Mauro Ciano, Leandro Gilioli

Facultad de Ingeniería – UBA

# Agenda

- ▶ **Introducción**
  - Motivación
  - Escenario
  - Problema a resolver
  - Soluciones existentes
- ▶ **Solución propuesta**
  - Búsqueda de la solución
  - Arquitectura de la solución
  - Tecnologías utilizadas
- ▶ **Demostración en vivo**
- ▶ **Cierre**
  - Posibles mejoras
  - Conclusiones



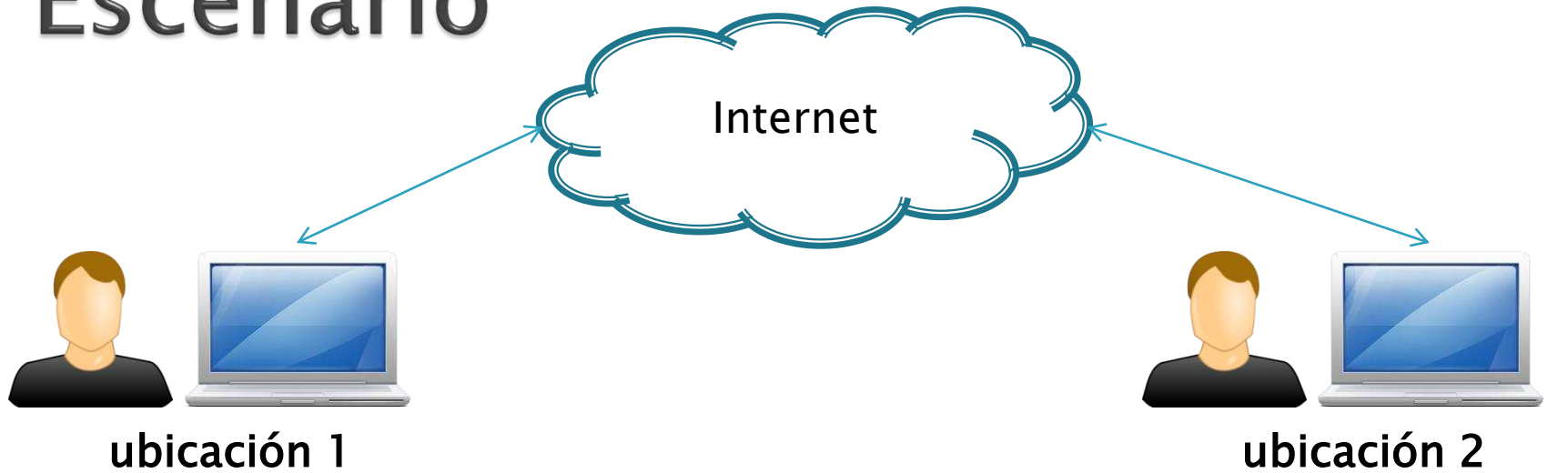
# Motivación



- ▶ ¿Cómo nace la idea del presente trabajo?
- ▶ Necesidad de trabajar sobre un mismo documento simultáneamente.
- ▶ Necesidad de obtener *feedback* instantáneo sobre tareas de desarrollo complejas.
- ▶ Varios desarrolladores trabajando sobre el mismo documento en diferentes lugares físicos.
- ▶ Herramientas existentes no satisfactorias.



# Escenario



```
class MyClass {  
    val value = 0;  
    // es difícil!  
}
```

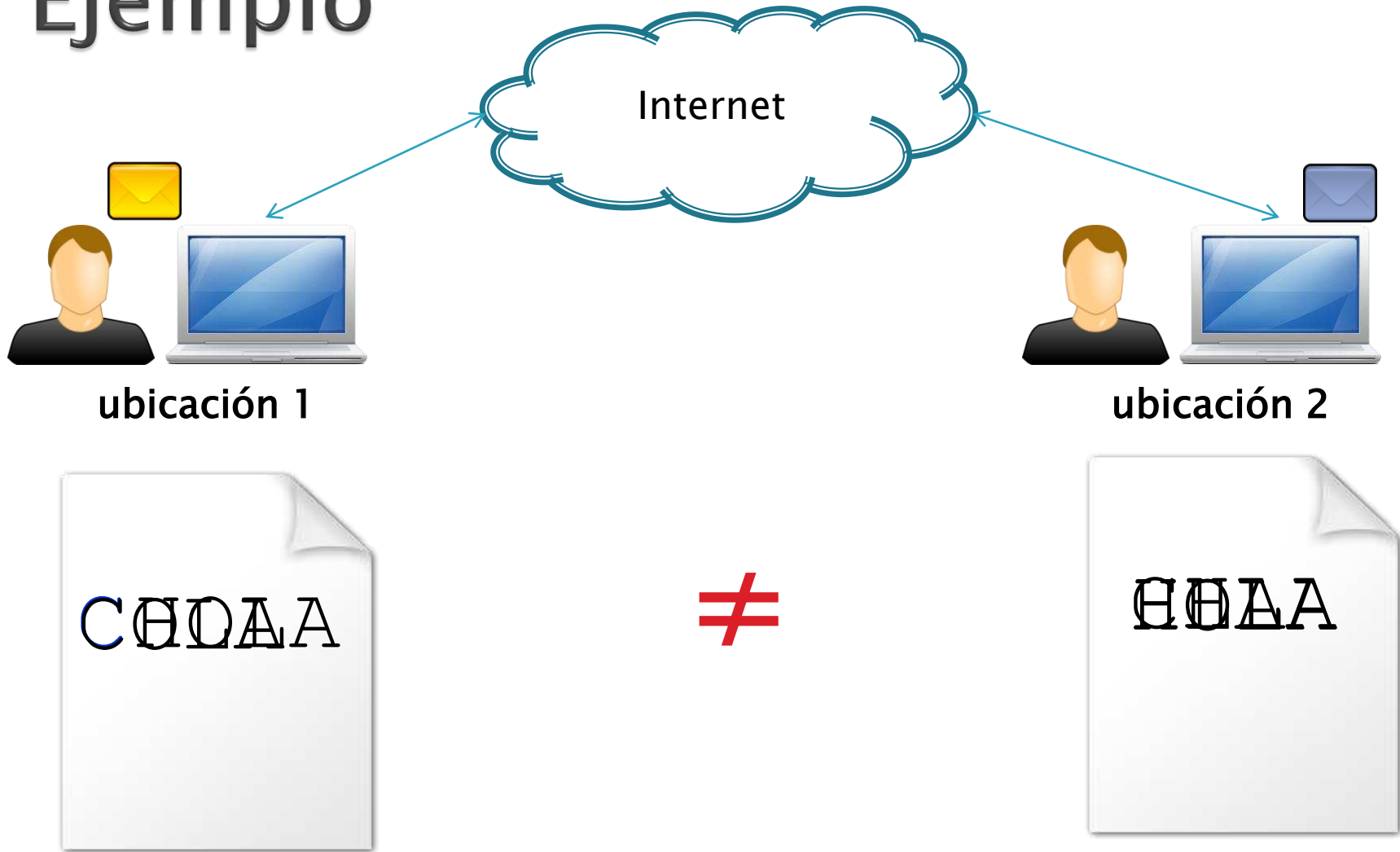
```
class MyClass {  
    val value = 0;  
    // es difícil!  
}
```

# Problemas a Resolver






- ▶ Mantener **sincronizado** el documento en todas las ubicaciones.
- ▶ Ofrecer un ***feedback*** instantáneo al usuario.
- ▶ No inventar otra herramienta, integrarse con las herramientas existentes.
- ▶ Simplicidad de uso y despliegue, sin necesidad de un servidor central.

# Ejemplo



# Soluciones existentes



Solución		Características	Puntos débiles
Google Docs		Edición de docs en tiempo real desde un navegador.	Solo utilizable a través de un browser. Código fuente cerrado.
Google Wave		Comunicación y colaboración en tiempo real.	Ídem Google Docs. El proyecto ha sido abandonado.
COLA (ECF)		Integración con Eclipse para colaboración en tiempo real de código fuente.	Máximo dos usuarios en una sesión de edición. Depende del proyecto ECF.
BeWeeVee		Framework para integración de funcionalidad de colaboración en tiempo real para .NET.	Código fuente cerrado. Está desarrollado sólo para la plataforma .NET.

# Búsqueda de la solución (I)



- ▶ Para sincronizar el documento entre cada ubicación se hace uso del algoritmo “Júpiter”.
- ▶ El algoritmo utiliza mensajes y operaciones:
  - Una operación modifica el estado de un documento.
  - Un mensaje contiene una operación e información de sincronismo.
- ▶ Al modificar un documento localmente se envían mensajes a las demás ubicaciones.
- ▶ Al recibir un mensaje remoto se transforma la operación y se la aplica localmente.

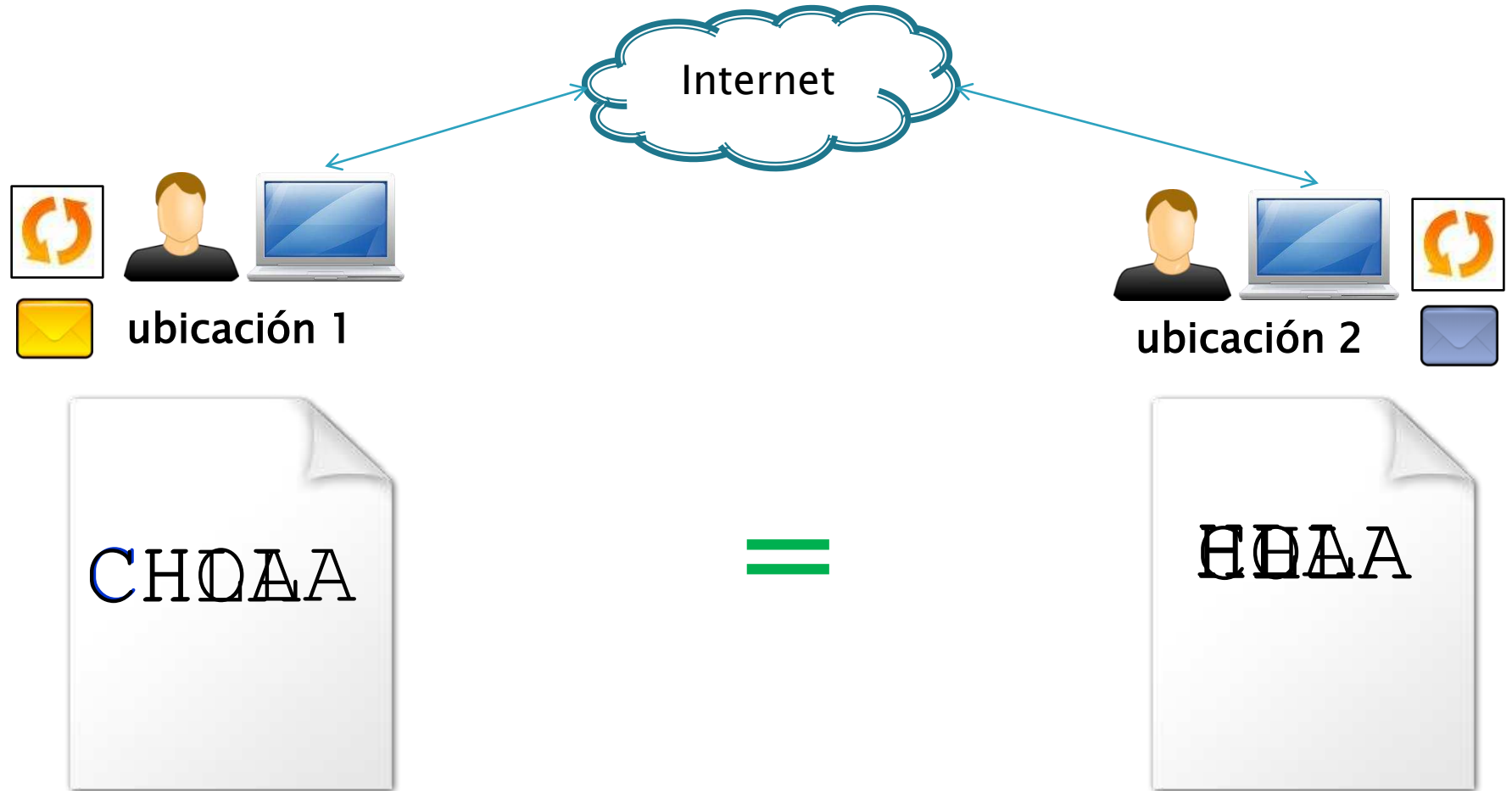
# Búsqueda de la solución (II)



- ▶ Originalmente se implementaron operaciones borrar e insertar de más de un carácter.
- ▶ Finalmente, se implementó la solución propuesta en el paper utilizado\* haciendo uso de operaciones borrar e insertar de un carácter de longitud.

\* Achieving Convergence With Operational Transformation in Distributed Groupware Systems.

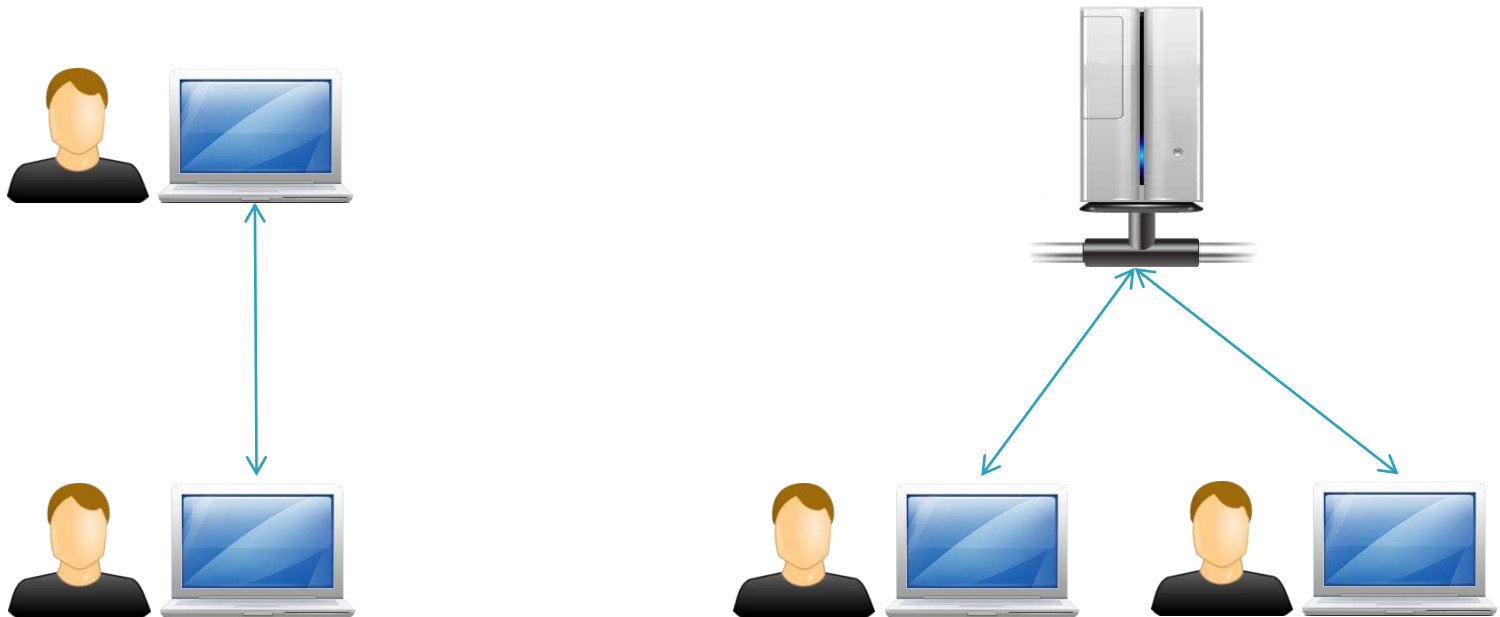
# Ejemplo utilizando sincronizadores



# Arquitectura de la solución



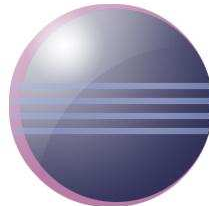
- Despliegue: Peer-To-Peer o Cliente-Servidor.





# Tecnologías y herramientas utilizadas

- ▶ Lenguajes de programación:  
Scala + Java.
- ▶ Eclipse IDE.
- ▶ IntelliJ IDEA.
- ▶ Maven.
- ▶ GIT.
- ▶ Spring.



**maven**

# Características de Scala



- ▶ Interoperabilidad con Java: Scala puede hacer uso de bibliotecas de Java y viceversa.
- ▶ Modelo basado Actores: reduce la complejidad en el desarrollo de aplicaciones concurrentes.
- ▶ Construcciones útiles:
  - Pattern matching.
  - Colecciones inmutables.
  - Funciones de orden superior (high-order functions).
  - Elementos del paradigma funcional.

# Metodología de desarrollo



- ▶ Scrum fue utilizado como metodología de desarrollo.
- ▶ El proyecto se concluyó en tres sprints.
- ▶ La duración de cada sprint fue de tres semanas.
- ▶ Luego de los tres sprint se generó la documentación.

# ¡Demostración en vivo!



Java EE - ParallelEditor-Eclipse/src/ar/nox/it/paralleeditor/eclipse/share/ShareManager.java - Eclipse

File Edit Source Refactor Navigate Search Project Run Window Help

ShareManager.java

```
154 » }  
155 »  
156 » @Override  
157 » public ISession connect(final ConnectionInfo info) {  
158 » » Assert.assertNotNull(info);  
159 »  
160 » » // connection id  
161 » » ConnectionId id = info.getId();  
162 » » // adapter  
163 » » DocumentsAdapter adapter = new DocumentsAdapter(converter, info);  
164 »  
165 » » // callback for username  
166 » » adapter.setOnLoginFailureCallback(new LoginFailureCallback(info, this));  
167 »  
168 » » // the newly created session  
169 » » JSession newSession = SessionFactory.newJSession(id.getHost(), id.getPort(), adapter);  
170 » » // log in to the kernel  
171 » » newSession.send(new RemoteLoginRequest(info.getUsername()));  
172 » » // store the remote session  
173 » » remoteSessions.put(info.getId(), newSession);
```

Chat

\*\* [18:41:05] User 'LeandroGilioli' has logged in.  
[18:42:31] LeandroGilioli said (from localhost:5000): Hi Mauro!  
[18:42:57] LeandroGilioli said (from localhost:5000): How're you doing?  
[18:43:11] you said (to localhost:5000): Great, this rocks!

Shares Console

Available hosts

- [X] localhost:5000 as MauroCiancio
- collab.servers.noxit.com.ar:5000 as MauroCiancio

Add hostname  
Delete hostname

Connection Details

Username: MauroCiancio  
Hostname: localhost  
Server Port: 5000

Hosting  
Stop

Available Users

- LeandroGilioli
- ▼ MauroCiancio
- /ParallelEditor-Eclipse/

Refresh docs and users

Available Docs

- /ParallelEditor-Eclipse/src/

localhost:5000 as MauroCiancio Send

Writable Smart Insert 166 : 81

# Posibles mejoras a implementar



- ▶ Mecanismo de deshacer cambios (*undo*).
- ▶ Esquema de autenticación y autorización.
- ▶ Integración con otros IDEs.
- ▶ Extensión de la solución a otros modelos.
- ▶ Cifrado de mensajes.

# Conclusiones



- ▶ Aplicaciones de naturaleza concurrente y distribuida precisan modelos que abstraigan al programador de la complejidad subyacente.
- ▶ Lenguajes nuevos sobre la JVM incorporan construcciones y paradigmas que simplifican el desarrollo.
- ▶ Los modelos para la implementación de sistemas colaborativos en tiempo real están en constante desarrollo y evolución.
- ▶ La colaboración en tiempo real permite interactuar a un equipo de trabajo de forma simple y coordinada acelerando, enriqueciendo y haciendo mas productivo su trabajo.

Preguntas?

# Muchas Gracias