VNIVERSIDAD NACIONAL AVFNOMA DE MEXICO

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

DISEÑO E IMPLEMENTACIÓN DE MECANISMOS DE CONTROL DE LA CONCURRENCIA PARA UN SISTEMA DE EDICIÓN COLABORATIVA EN INTERNET E INTRANET

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRA EN CIENCIAS (COMPUTACIÓN)

P R E S E N T A:

SILVIA RAMIREZ GARCIA

DIRECTOR DE LA TESIS: DR. MANUEL ROMERO SALCEDO

MÉXICO, D.F. **2003.**

A mi esposo Fidel

A mis padres Silvia y Hermilo

A mi hermano Fabián

B

Agradecimientos

A mi director de tesis el Dr. Manuel Romero Salcedo y al M. Carlos Rodríguez Contreras, por el apoyo y la aportación de valiosas contribuciones para la elaboración de esta investigación.

A los doctores Hanna Oktaba, Amparo López Gaona y Gerardo Vega Hernández por sus importantes aportaciones y sugerencias para la realización de este trabajo de tesis.

A mis compañeros de maestría, el Ing. César Murillo y el Ing. Aníbal Avelar Rosales, por su ayuda y colaboración en este proyecto.

Al Instituto de Investigación en Matemáticas Aplicadas y en Sistemas (IIMAS) y al Consejo Nacional de Ciencia y Tecnología (CONACY) por los recursos facilitados para la realización de mis estudios.

Índice

E

Prefacio

Aunque la tarea de edición de texto ha sido considerada como un esfuerzo individual en la que los escritores trabajan de forma aislada, la mayor parte de ésta implica interacción social. Tanto en la industria como en la academia, la tarea de edición es realizada, en gran medida, por grupos de personas. Para apoyar la edición de escritura electrónica entre personas que trabajan en conjunto, se ha desarrollado una disciplina denominada ***Edición Colaborativa Asistida por Computadora*** (***CSCWriting***[1]), la cual ha aportado numerosas contribuciones enfocadas al desarrollo de sistemas y herramientas que facilitan la producción de documentos en forma colaborativa. Esta disciplina se ha consolidado como un nuevo tema de investigación en la tecnología de la información. Esto se debe a que, por un lado varios trabajos de investigación han reportado que la mayoría de los documentos que se producen son elaborados a través de un esfuerzo colaborativo y, por otro lado, se ha demostrado que varios autores en grupos pueden producir documentos de mejor calidad gracias a la confrontación de sus experiencias, a la combinación de sus conocimientos, a sus niveles de especialización y a la contribución que cada uno puede ofrecer.

La mayoría de las investigaciones realizadas sobre el proceso de escritura, han afirmado que esta actividad involucra más que palabras, es una forma de plasmar las ideas elaboradas por la mente, en la que intervienen la planeación, la producción de ideas, la revisión del lenguaje y la coordinación motriz. Con respecto a las necesidades de los colaboradores en un sistema de edición, se han definido algunas categorías concernientes a los diferentes componentes que intervienen en el proceso de escritura: métodos de control de documentos, estrategias de edición, roles de los colaboradores, actividades del proceso de edición, entre otros.

Actualmente, existen diferentes propuestas de sistemas de edición colaborativa como lo son GROVE, PREP, SASSE, MACE, Alliance, etc. Cada uno de ellos se distingue por los mecanismos utilizados en el proceso de edición y organización de tareas (planeación, lluvia de ideas, escritura, consolidación). El desarrollo de sistemas de edición colaborativa continúa incorporando nuevas tecnologías (lenguajes, arquitecturas, algoritmos y hardware) con el propósito de mejorar la comunicación, la colaboración y la coordinación de los participantes en la elaboración de documentos. Así también, se auxilia de las investigaciones realizadas en áreas como CSCW[2], groupware y sistemas distribuidos.

[1] Del inglés *"Computer Supported Cooperative Writing"*
[2] Del inglés *"ComputerSupported Cooperative Work"*

Antecedentes de colaboración

El presente trabajo forma parte del diseño del ***sistema de Edición coLaborativa de documentos XML en Internet (ELXI).*** El objetivo de este sistema es la elaboración de documentos de cualquier índole (tesis, revistas, columnas periodísticas o libros), a través de la colaboración de autores y lectores, situados en lugares que pueden estar geográficamente distantes. Una de las aportaciones de este sistema es el manejo de la información en documentos estructurados mediante el lenguaje de marcado extensible (XML[3]), lo que permite que el manejo de cada archivo sea más dinámico.
El diseño de este sistema se ha basado principalmente en la característica de permitir que existan varios escritores editando un documento a la vez. Aunque, el sistema todavía no es un producto final, se caracteriza por su funcionalidad debido a que parte de la arquitectura, la administración de los documentos, el espacio de trabajo y la sesión de trabajo han sido implementados exitosamente.
La arquitectura base de ELXI fue construida con la colaboración del Ing. César Murillo, Ing. Aníbal Avelar Rosales y la que suscribe. Cada uno de nosotros ha intervenido en el desarrollo de este sistema con un tema en específico:

- Estructura de documentos XML
- Consistencia sobre los documentos XML replicados
- Concurrencia sobre los documentos XML

El sistema ELXI sigue en constante desarrollo, y ha despertado el interés de algunos grupos de trabajo de edición electrónica. Asimismo, ha sido impulsado, por un lado, por el Consejo Nacional de Ciencia y Tecnología (CONACYT) bajo el proyecto "Ambiente Multipropósito para la Edición Colaborativa en Intranet e Internet" con referencia 4003165-J3204A, dirigido por el Dr. Manuel Romero Salcedo; y por el otro lado, por el Instituto Mexicano del Petróleo bajo el proyecto "Computación Distribuida Inteligente" (CDI) D.00006, en el cual el Dr. Manuel Romero participa.

Objetivo

El principal objetivo de este trabajo de tesis es implementar un mecanismo de control de concurrencia al sistema ELXI. De tal manera que prevengan o resuelvan los problemas de inconsistencia de información, que puedan surgir cuando múltiples autores escriban al mismo tiempo en un mismo documento. Tomando en cuenta que ELXI permite a cada usuario trabajar de manera remota, se han considerado ciertos aspectos para el desarrollo del mecanismo propuesto. Entre ellos, se tiene que los documentos originales deben residir en una máquina llamada servidor, y que las aportaciones de cada uno de los autores deben ser visibles a todos los participantes que trabajan en el mismo documento.

[3] Del inglés *"Extensible Markup Lenguaje"*

Para implementar el mecanismo de control de concurrencia, se ha realizado una investigación a temas propios de los sistemas colaborativos, así como de los sistemas distribuidos.

Organización

Este trabajo de tesis consiste en cuatro capítulos. El primero da una argumentación del porque el campo de investigación de los sistemas distribuidos es importante en el desarrollo de los sistemas colaborativos que asisten las actividades y el trabajo en grupo. Este capítulo también resalta el problema de la inconsistencia de la información cuando existe un ambiente multiusuario. Asimismo se exponen algunos mecanismos de control de concurrencia para resolver dicho problema.
El capítulo dos está enfocado en el estado del arte de los sistemas de trabajo colaborativo asistido por computadora. Se describe la evolución de los sistemas groupware, así como, los elementos necesarios para su diseño. También se citan algunos ejemplos de los sistemas de edición colaborativa, señalando los mecanismos que incorporan para controlar el proceso de edición concurrente.
El capítulo tres describe cuál es el problema de control de concurrencia en la edición de documentos en el sistema de autoría colaborativa ELXI. Asimismo, se realiza una propuesta para su solución. Para la integración del mecanismo de control de concurrencia se realiza un análisis del estado actual del sistema ELXI, además de las necesidades a cubrir. Posteriormente, se propone un diseño descrito mediante diagramas UML[4]. Por último, se explican algunos fragmentos de código del mecanismo implementado.
En el capítulo cuatro se exponen los resultados que se obtuvieron al hacer las pruebas de funcionalidad sobre el mecanismo de control de concurrencia. En estas pruebas intervinieron usuarios reales cuyo objetivo fue el de realizar un documento. De igual manera, se realizaron pruebas de desempeño, en las cuales se evaluaron dos esquemas concernientes al almacenamiento de la información de los documentos.
Finalmente, en el capítulo cinco se ofrecen las conclusiones sobre el trabajo realizado y se hace mención de la experiencia personal que se experimentó al trabajar en forma colaborativa.

[4] Del inglés *"Unified Model Lenguage"*

Capítulo 1

Importancia de los sistemas distribuidos en los sistemas de trabajo colaborativo

Con el surgimiento de nuevas tecnologías en las comunicaciones y en el campo de la computación, se han ido optimizando los sistemas de control de información. Principalmente, se han salvado distancias y minimizado el tiempo para la obtención de datos. Debido a ello, la demanda de los sistemas de cómputo se ha incrementado, así como también las necesidades de adaptación de los usuarios finales para interaccionar con ellos.

Algunas de las aplicaciones que han surgido dentro de estas tendencias, como es el caso del correo electrónico, los cuartos de conversación[5], la transferencia de archivos, la videoconferencia, etc., han facilitado el desarrollo de algunas actividades llevadas a cabo por personas que trabajan de manera conjunta para resolver un problema en común. Sin embargo, estas aplicaciones no han cubierto del todo las necesidades de los grupos de trabajo, debido a que no proporcionan las suficientes herramientas para apoyar las tareas de comunicación, colaboración y coordinación de actividades. Para resolver esta problemática, en el área de investigación de ***Trabajo Colaborativo Asistido por Computadora (CSCW),*** junto con el de ***Interacción Humano Computadora (HCI*** [6]***)***, se han desarrollado sistemas colaborativos que asisten las actividades y el trabajo en grupo. Sin embargo, debido a que los usuarios frecuentemente se encuentran distribuidos en tiempo o espacio, es importante considerar mecanismos que faciliten el almacenamiento de la información compartida y consistente, la coordinación del trabajo y la comunicación a través de medios eficientes. Estos mecanismos han sido desarrollados dentro del área de los sistemas distribuidos, y se han convertido en parte del campo de investigación de CSCW. Considerando la importancia de esto, en las siguientes secciones se exponen las arquitecturas más comunes que se han implementado en los sistemas distribuidos. Asimismo, se describe la estrategia de programación que es utilizada para atender las solicitudes de múltiples usuarios en un sistema. De igual manera, se explican algunos de los mecanismos que existen para resolver problemas de inconsistencia de información y alteración en las operaciones ejecutadas que surgen en ambientes multiusuarios. Inclusive, se hace un análisis de estos mecanismo mencionando sus ventajas y desventajas, que será utilizado en la parte en la que se realiza la propuesta de este trabajo de tesis.

[5] Del inglés *"Chats"*
[6] Del inglés *"Human Computer Interaction"*

1.1 Sistemas groupware

Para apoyar las actividades que realizan un grupo de personas en la realización de una tarea en común, se deben considerar tres aspectos clave: la comunicación, la colaboración y la coordinación. La comunicación es uno de los aspectos más importantes en un grupo de trabajo, ya que mediante ella se realiza el intercambio de información necesaria para el desarrollo de una actividad. Ésta puede ser espontánea y sin limitantes cuando las personas se encuentran en un mismo lugar frente a frente[7]. Sin embargo, cuando se encuentran a distancia pueden recurrir a diferentes medios, por ejemplo el teléfono, aplicaciones creadas para Internet como los cuartos de conversación, o bien, a medios más sofisticados: los sistemas de videoconferencia. Hay aplicaciones, en las que no se permite una interacción entre usuarios, tan sólo realizan el envío de mensajes, como el correo electrónico.

Otro pilar en las actividades de grupo es la colaboración, en la cual se establece la relación entre el trabajo individual y de grupo. Además, para hacerla efectiva es necesaria la compartición de información. Algunos de los grupos de trabajo se auxilian de los sistemas de cómputo como las bases de datos o los servidores de archivos para que múltiples usuarios puedan acceder a un mismo conjunto de datos.

La comunicación y la colaboración pueden ser mejoradas si las actividades del grupo son coordinadas [Ell91]. Por ejemplo, cuando dos usuarios alteran un mismo documento al mismo tiempo, cada quien tendrá una versión del documento, siendo inconsistente la información para ambos. Una posible solución a este problema es la edición del documento por turnos, una vez que un colaborador haya finalizado con las modificaciones pertinentes, el siguiente colaborador podrá trabajar en el documento. Cuando este último haya terminado, el primer usuario podrá disponer de la versión del segundo. Este tipo de coordinación permite que ambos usuarios obtengan las aportaciones que uno y otro ha escrito en el mismo documento.

Se puede apreciar que los grupos de personas se están apoyando en los sistemas de cómputo para llevar a cabo las actividades de comunicación, colaboración y coordinación. Sin embargo éstas no han podido ser apoyadas plenamente debido a que la mayoría de estos sistemas proporcionan una interacción entre usuario y sistema, y no la incorporan entre usuarios. Debido a esto, ha surgido la necesidad del diseño de sistemas de cómputo que asistan el trabajo en grupo, mejor conocidos como sistemas groupware. Dichos sistemas, se han enfocado en la solución de los problemas de la comunicación y de la coordinación de actividades en la realización de una tarea en común. Ellis [Ell91] propone la siguiente definición de los sistemas groupware:

"Sistemas de cómputo que asisten a un grupo de personas involucradas en una tarea o propósito en común y que proveen una interfaz a un espacio compartido".

En esta definición, hay dos aspectos importantes que subrayar tarea o propósito en común y espacio compartido, por lo que quedan excluidos aquellos sistemas multiusuarios, en donde los usuarios no comparten alguna tarea en común.

Con respecto al espacio de trabajo compartido representado por computadoras, éste consiste en una simulación de espacio de trabajo físico (un escritorio, un salón, una oficina, etc.), en donde las personas pueden ver y manipular los objetos relacionados con las actividades que realizan. El

[7] Del inglés *"face-to-face"*

término compartido hace énfasis en que estos objetos pueden ser usados en forma simultánea por todas las personas que interactúan dentro del espacio de trabajo.
Ellis propone que la interacción en el trabajo colaborativo puede clasificarse en las dimensiones de tiempo y espacio. En la figura 1.1 se muestra esta división.

	Mismo Tiempo	**Diferente Tiempo**
Mismo lugar	Interacción frente a frente	Interacción asíncrona
Diferente lugar	Interacción síncrona distribuida	Interacción asíncrona distribuida

Figura 1.1 Tabla de clasificación "Tiempo y Espacio" [Ell99].

La interacción síncrona se refiere a que el grupo de personas están interactuando al mismo tiempo y dependiendo del lugar que compartan puede ser frente a frente o distribuidos. En la interacción asíncrona las personas interaccionan en diferentes tiempos y dependiendo del lugar que compartan es local o distribuida. En la figura 1.2, se muestran algunos ejemplos de herramientas colaborativas para cada tipo de interacción.

	Mismo Tiempo	**Diferente Tiempo**
Mismo espacio	Apoyo a la toma de decisiones Reuniones electrónicas frente a frente Pizarrones electrónicos Edición colaborativa	Pizarrón de notas Salones virtuales Hipertexto Colaborativo Calendarios de grupo
Diferente espacio	Teléfono Cuartos de conversación Videoconferencia Edición colaborativa	Correo electrónico Boletín de noticias Agenda electrónica Edición colaborativa

Figura 1.2 Ejemplos de tipos de interacción.

Las herramientas colaborativas que brindan a los participantes una interacción en diferentes instantes de tiempo son conocidas como de tiempo definido o ***asíncronas***. Aquellas herramientas que permiten la interacción de los participantes en un mismo tiempo son llamadas de tiempo real o ***síncronas***. Algunos sistemas groupware como DeskTOP [Por99], NCS Habanero [WNHa] y TeamWave [WTeW] proporcionan herramientas síncronas, por ejemplo cuartos de conversación, pizarrón blanco y votación. Pero también existen sistemas que sólo incluyen herramientas asíncronas como la agenda, el administrador de documentos y notas de discusión, este es el caso de BSCW[8] [WBSC].
Como anteriormente se mencionó, el diseño de estos sistemas colaborativos se apoya en mecanismos propios de los sistemas distribuidos, es por ello que a continuación se revisan las arquitecturas más comunes que se pueden encontrar en la literatura.

1.2 Sistemas distribuidos

Un sistema distribuido es aquel en el que los componentes de hardware y software localizados en una red de computadoras, se comunican y coordinan sus actividades mediante el paso de mensajes [Cou01]. Esto se puede ver como una colección de computadoras autónomas enlazadas

[8]Del inglés *"Basic Support for Collaborative Work"*

por una red y equipadas con el hardware y software necesario para permitir que los trabajos de los clientes se lleven a cabo (ver figura 1.3). Ejemplos de sistemas distribuidos son la Internet que es una colección de redes de computadoras interconectadas de diferentes tipos. Asimismo, las intranets que son un subconjunto de la Internet, tienen su propia administración, configuración y políticas de seguridad. Otro ejemplo de sistema distribuido es la World Wide Web, que permite la publicación y el acceso a recursos y servicios a través de la Internet.

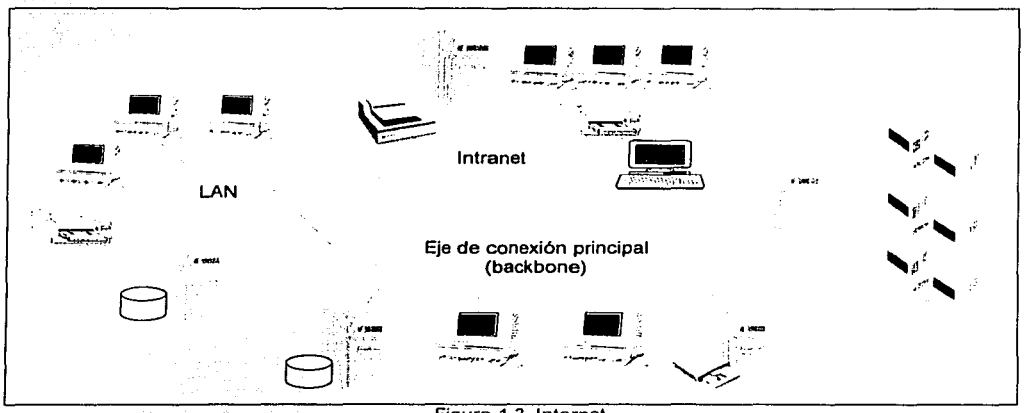


Figura 1.3. Internet.

En la construcción de los sistemas distribuidos se tiene diferentes aspectos que cubrir, entre ellos se encuentran:

- **Heterogeneidad**. Se refiere a la variedad de redes, componentes de hardware, sistemas operativos, lenguajes de programación y aplicaciones que pueden coexistir en un sistema. En el caso de Internet, existen diferentes tipos de redes, sin embargo la comunicación entre ellas es posible gracias a que adoptan un mismo protocolo de comunicación. Para resolver el problema de heterogeneidad, otros sistemas han implementado una capa de software llamada *mediador*[9], para permitir la comunicación entre recursos sin problemas de compatibilidad.
- **Abierto**. En los sistemas distribuidos esta propiedad está determinada por el grado de recursos que pueden ser agregados y a la vez disponibles para el uso de una variedad de programas cliente. Sin embargo, también se requiere de la especificación y documentación del sistema para llevar a cabo su extensión o re-implementación.
- **Seguridad**. Los recursos de información tienen un alto valor para los usuarios y es por eso que en la actualidad se considera de gran importancia su seguridad. En cualquier sistema, sea o no distribuido, se deben considerar tres componentes importantes en los recursos de información y estos son la confidencialidad (protección contra individuos no autorizados), la integridad (protección contra alteración o corrupción) y la disponibilidad (protección contra interferencia en el acceso de recursos).
- **Escalabilidad**. Un sistema es escalable si su efectividad permanece cuando se ha incrementado de manera significante el número de recursos y clientes. La Internet es un

[9] Del inglés "*middleware*"

ejemplo claro de un sistema escalable debido al crecimiento inmediato del número de computadoras y servicios. Sin embargo, el diseño de un sistema distribuido presenta algunos desafíos como controlar el costo de recursos físicos, controlar la pérdida de rendimiento, prevenir que la efectividad de los recursos de software no se agote, evitar los cuellos de botella.

- **Manejo de fallas**. Cuando ocurren fallas de hardware o software es de esperarse que los programas produzcan resultados incorrectos, o bien, no finalicen el cálculo solicitado. Una ventaja que se busca en los sistemas distribuidos es que las fallas sean parciales, es decir, si un componente falla los demás continúan funcionando. Las fallas pueden ser detectadas tanto en los clientes como en los servidores, para resolver el problema o enmascaradas para hacerlas menos severas. Por ejemplo, cuando un buscador de páginas no puede contactarse a un servidor Web, le informa al usuario acerca del problema. Los servicios pueden ser tolerantes a fallas mediante el uso redundante de componentes, por ejemplo, en el Sistema de Nombre de Dominio (DNS[10]), cada tabla de nombres es replicada al menos en dos diferentes servidores.
- **Concurrencia**. En un sistema distribuido es importante que se atienda la petición de un cliente, al igual que las peticiones de múltiples clientes sobre un mismo recurso, de tal forma que no sea negado el servicio solicitado a ninguno de ellos. Para que un recurso sea concurrente, se ha empleado el uso de procesos ligeros, en donde cada uno de ellos atenderá las solicitudes de los clientes, dando la impresión de que son ejecutadas al mismo tiempo.
- **Transparencia**. Mediante esta característica un sistema distribuido es percibido como un todo a pesar de que está constituido por múltiples componentes independientes. Es decir, el usuario final no percibe que los componentes se encuentran separados en el sistema distribuido gracias a la aplicación que los oculta.

1.2.1 Arquitecturas de sistemas distribuidos

Otro aspecto importante en el diseño de los sistemas distribuidos es la asignación de tareas para cada uno de los componentes que lo integran, por lo que se han creado diferentes arquitecturas, de las cuales, las más comunes se mencionan a continuación.

Modelo cliente-servidor. En esta arquitectura existe, por un lado, un conjunto de procesos[11] servidores, cada uno actuando como un manejador de recursos de un determinado tipo, y por otro lado, existe también una colección de procesos clientes, cada uno ejecutando una tarea que requiere el acceso a algún recurso de hardware y software compartido (ver figura 1.4.a). Los servidores también pueden llegar a ser clientes de otros servidores. Por ejemplo, un servidor Web es frecuentemente un cliente de un servidor de archivos local que administra los archivos, en los cuales, las páginas Web son almacenados.

Servicios proporcionados por múltiples servidores. Esta arquitectura consiste de varios procesos servidores que se ejecutan en diferentes computadoras y cada uno de ellos se coordina con los demás para proporcionar un servicio a los procesos cliente (ver figura 1.4.b). Un ejemplo

[10] Del inglés *"Domain Name System"*
[11] Proceso es una instancia de un programa en ejecución

de este tipo de arquitectura es la Web, cada servidor administra su propio conjunto de recursos y cada usuario mediante un buscador de páginas puede acceder a un recurso que se encuentre en cualquiera de esos servidores. Otro ejemplo es la replicación, la cual proporciona múltiples copias de datos consistentes a través de procesos ejecutados en diferentes computadoras. Los principales beneficios que se busca de ella es el incremento del rendimiento, la disponibilidad y mejorar la tolerancia a fallas.

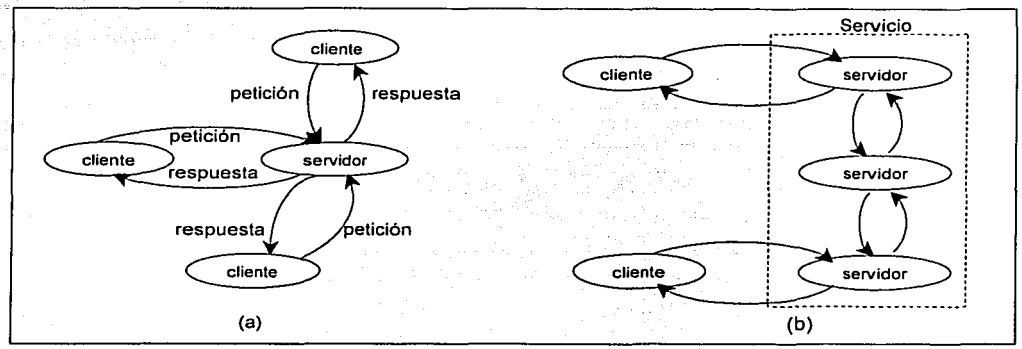


Figura 1.4. (a) Modelo cliente-servidor (b). Servicios proporcionados por múltiples servidores.

Servidores intermediarios de Web[12] y memoria temporal. La memoria temporal es un medio de almacenamiento de objetos de datos usados recientemente. Cuando un objeto es necesitado por un proceso cliente el servicio de memoria temporal revisa si esta disponible una copia del objeto, en caso contrario es buscado, copiado y almacenado a este medio. La memoria temporal puede ser colocada en cada cliente o en un servidor intermediario de Web compartida por diversos clientes (ver figura 1.5.a). Por ejemplo, mediante una petición HTTP se revisa que las copias de las páginas almacenadas en la memorial temporal del buscador estén actualizadas con respecto a las originales que residen en el servidor antes de ser desplegadas.

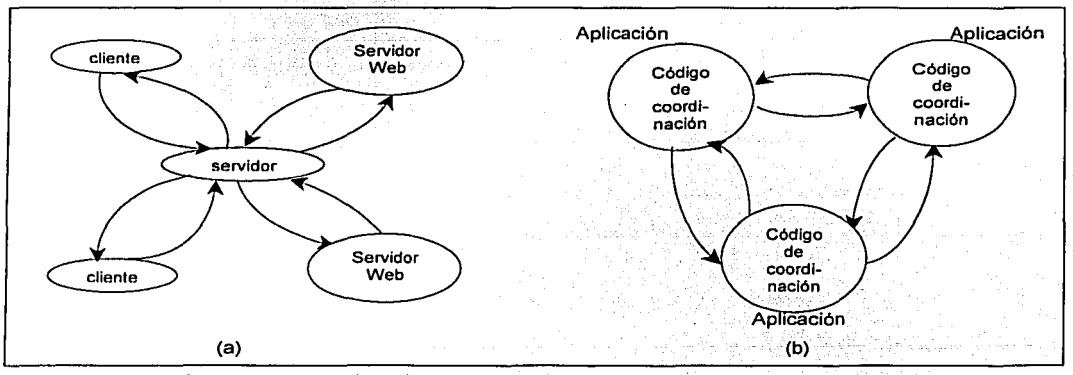


Figura 1.5 (a) Servidores intermediarios de Web y cache. (b) Par de procesos.

[12] Del inglés *proxy*

Procesos pares. En esta arquitectura todos los procesos interactúan cooperativamente uno con otro (como un par) para ejecutar una actividad distribuida sin distinguir entre clientes y servidores. En este modelo, los procesos pares mantienen la consistencia de los recursos a nivel aplicación y sincronizan acciones a nivel aplicación cuando es necesario. En la figura 1.5.b, los procesos pares pueden interactuar uno con otro y la comunicación dependerá de los requerimientos de la aplicación. La ventaja que ofrece este modelo, al eliminar los procesos servidor, es la reducción de retardos en la comunicación de objetos locales. Por ejemplo, consideremos la aplicación del pizarrón blanco distribuido, el cual permite que usuarios en diferentes computadoras vean y modifiquen interactivamente una imagen que es compartida entre ellos. Esta puede ser implementada como un proceso aplicación en cada sitio que depende del mediador, para ejecutar el evento de notificación y comunicación de grupo, y avisar a todos los procesos aplicación los cambios en la imagen.

1.2.2 Construcción de aplicaciones distribuidas

La parte principal de la arquitectura de las aplicaciones distribuidas es el mediador, el cual es una capa de software cuyo propósito es enmascarar la heterogeneidad. El mediador está representado por procesos u objetos que interactúan entre un conjunto de computadoras con el propósito de apoyar la comunicación y la compartición de recursos en las aplicaciones distribuidas. En particular, incrementa el nivel de comunicación de las actividades de los programas a través del soporte de abstracciones, tales como: la invocación del método remoto, la notificación de eventos, replicación de datos compartidos y la transmisión de datos multimedia en tiempo real. Actualmente, existen diferentes modelos para diseñar el mediador de una aplicación distribuida, entre los cuales se encuentra el modelo de *llamadas a procedimientos remotos* (RPC[13]), el modelo de *invocación de métodos remotos* (RMI[14]) y el modelo de *programación orientada a eventos*.
El modelo más conocido es el de llamadas a procedimientos remotos, el cual permite que los programas clientes llamen a los procedimientos remotos de programas servidores, que se ejecutan en procesos separados y generalmente en diferentes máquinas. El mediador que permite la implementación de este modelo es el modelo de objeto de componente distribuido (DCOM[15]) [WDco], el cual proporciona una estructura para desarrollar sistemas distribuidos basados en objetos. Por ejemplo, el servidor de transacciones de Microsoft (MTS[16]) esta construido con la tecnología DCOM principalmente para la comunicación con administradores de recursos como SQL Server u otros servidores MTS. No obstante, la desventaja principal de esta tecnología es que sólo se orienta a aplicaciones Windows.
Otro mediador que implementa el modelo RPC es el Sun RPC. Casi todos los sistemas UNIX lo utilizan principalmente para implementar el sistema de archivos en red (NFS[17]), el cual es un sistema de archivos distribuidos que ha sido recomendado como un estándar en Internet.
El modelo más reciente, y totalmente orientado a objetos, es el de invocación a métodos remotos. Este modelo es una extensión de la invocación de métodos locales y permite a los objetos que se están ejecutando en una computadora invocar a los métodos de objetos que se encuentran en

[13] Del inglés *"Remote Procedure Call"*.
[14] Del inglés *"Remote Methode Invocation"*.
[15] Del inglés *"Microsoft's Distributed Common Object Model"*.
[16] Del inglés *"Microsoft's Transaction Server"*.
[17] Del inglés *"Network File System"*.

otras computadoras. Estos métodos se ejecutan en computadoras remotas, de ahí el nombre invocación remota de métodos. Los objetos invocados remotamente realizan cálculos y pueden devolver valores que son utilizados por los objetos locales. Java tiene una API[18] que permite implementar este modelo llamado Java RMI [WJav]. Con este modelo trabaja Netsurface [WNet], un sistema orientado a apoyar las actividades del mundo de los negocios, trabajo en equipo y compartición de la información en una Intranet corporativa.
Dentro del modelo RMI también se encuentra la arquitectura de intermediación de solicitud de objetos comunes (CORBA[19]) [WCOR], admite una estructura orientada a objetos para el desarrollo de sistemas distribuidos. El punto fuerte de CORBA es que admite un modelo del lenguaje independiente. Sin embargo, esta ventaja se convierte en una desventaja para Java, ya que obliga a sus aplicaciones utilizar un modelo de objetos externo. Un ejemplo de un sistema que incorpora una arquitectura CORBA es COLA[20] [Tre94], el cual es utilizado para el desarrollo de aplicaciones colaborativas síncronas.
Por último, el modelo de *programación orientado a eventos* se basa en el envío de mensajes generados a partir de los eventos de un objeto. Algunos ejemplos de eventos de un sistema son la pulsación de una tecla o del ratón, el desplazamiento del ratón o el ajuste del tamaño de una ventana. Debido a la demanda de aplicaciones distribuidas, este modelo se ha extendido para poder escribir programas basados en objetos distribuidos. Un ejemplo de ello, es Mushroom [WMus], cuyo objetivo es apoyar la colaboración y la interacción de grupo en Internet. Este sistema proporciona espacios de trabajo que contienen representaciones de los usuarios que están presentes, incluyendo la información de los objetos compartidos como documentos, presentaciones multimedia y pizarrones blancos. Para actualizar la vista del espacio de trabajo se hace uso de eventos, los cuales son disparados según la acción del usuario causando la distribución de mensajes al servidor y a los clientes activos.

1.2.3 Procesos e hilos

El término proceso fue utilizado por primera vez por los diseñadores del sistema Multics en los años sesenta. Desde entonces, el término proceso, utilizado a veces como sinónimo de tarea, ha tenido diferentes definiciones. La más aceptada es la de abstracción de un programa en ejecución, compuesto por el código ejecutable, una sección de datos que contienen las variables globales, una sección de pila que contiene datos temporales (parámetros de subrutinas, direcciones de retorno y variables temporales) y un estado de los registros del procesador [Tan98].
En la mayoría de los sistemas operativos, la esencia de un proceso la constituyen dos conceptos independientes [Bac92]:

- La unidad de ejecución (unidad del programador de tareas en el sistema operativo)
- La memoria y otros recursos con los cuales la unidad de ejecución esta asociada.

Haciendo referencia al primer concepto, un proceso es considerado como aquella unidad a la que se le asigna un espacio de memoria u otros recursos como dispositivos de E/S. Este tipo de proceso también es conocido como proceso pesado, el cual tiene definido su propio espacio de

[18] Del inglés *"Aplication Programming Interface"*.
[19] Del inglés *"Common Object Request Broker Arquitecture"*.
[20] Del inglés *"Cooperating Objects in Lightweight Activities"*.

memoria. Al realizar el sistema operativo un cambio de contexto[21] entre procesos, éste resulta muy alto ya que mientras se coloca en memoria el proceso a ejecutarse, hay que poner en espera el que se estaba ejecutando.
Con respecto al segundo concepto, un proceso es una instancia de ejecución de un programa que comparte espacios de memoria u otros recursos, por ejemplo archivos. Tales procesos son frecuentemente llamados procesos ligeros o hilos. Este término es usado para indicar que el cambio de contexto es bajo, es decir, el sistema operativo no necesita cambiar la administración de memoria u otro recurso de información para ejecutar un nuevo hilo.
El uso de los hilos ha sido de gran utilidad entre servidores, ya que ha reducido la tendencia a que los mismos servidores se conviertan en cuellos de botella al procesar concurrentemente múltiples solicitudes cliente. Así, un hilo puede procesar la petición de un cliente, mientras un segundo hilo sirve otra petición que espera el acceso a un disco. Existen diferente arquitecturas de servidores multi-hilos y a continuación se describen las más comunes.

Arquitectura de arreglo de trabajadores. El servidor crea un arreglo de hilos trabajadores para procesar las peticiones cuando inicie. En la figura 1.6, el módulo "Recibir y Encolar" está implementado por un hilo de E/S, el cual recibe peticiones de una colección de puertos y los sitúa en una cola de petición compartida para ser atendido por los trabajadores. Una desventaja de esta arquitectura es que no es flexible, el número de trabajdores en el estanque puede ser muy pequeño para atender el promedio actual de peticiones.

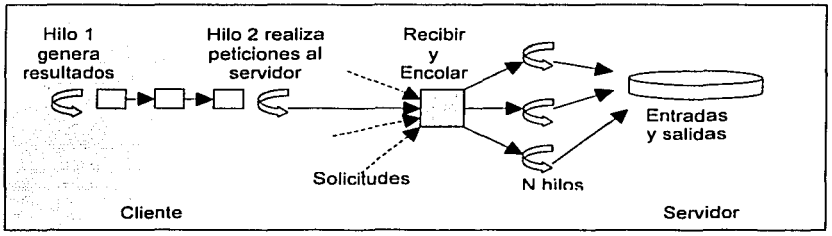


Figura 1.6. Cliente y servidor con hilos.

Arquitectura hilo por petición. El hilo de entrada y salida (E/S) crea un hilo trabajador por petición, y este hilo trabajador se destruye cuando se ha procesado la petición. La ventaja de esta arquitectura es que los hilos no compiten por su posición en una lista de espera y además, el hilo E/S puede crear rnuchos trabajadores como peticiones existan. Sin embargo, esta situación genera la sobrecarga de creación de hilos y destrucción de operaciones (ver figura 1.7.a).

Arquitectura de hilo por conexión. Asocia un hilo con cada conexión. El servidor crea un nuevo hilo trabajador cuando un cliente hace una conexión y destruye el hilo cuando el cliente cierra la conexión. El cliente puede hacer muchas peticiones sobre la conexión establecida con el servidor, dirigiéndose a uno o más objetos (ver figura 1.7.b).

[21] El cambio de contexto es la operación que consiste en desalojar a un proceso de la CPU y reanudar otro. Esto implica guardar el estado del proceso que sale en su BCP(bloque de control de proceso), y recuperar los registros del proceso que entra.

Arquitectura hilo por objeto. Asocia un hilo con cada objeto remoto. Un hilo I/O recibe peticiones y las encola para los trabajadores, pero esta vez existe una cola por objeto (ver figura 1.7.c).

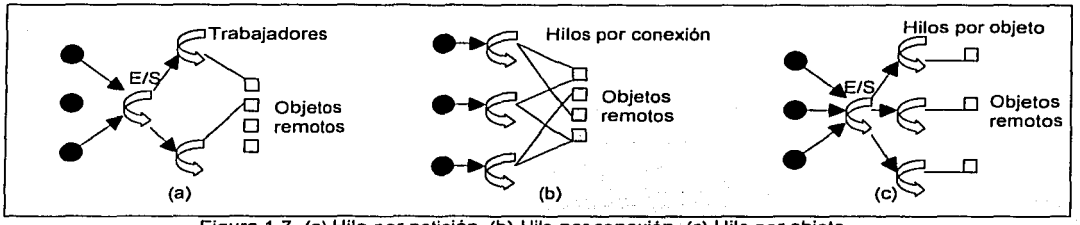


Figura 1.7. (a) Hilo por petición. (b) Hilo por conexión. (c) Hilo por objeto.

En las arquitecturas de hilo por conexión e hilo por objeto, el servidor crea un menor número de hilos en comparación con la arquitectura hilo por petición. Sin embargo, una desventaja es que los clientes pueden retardarse más en su respuesta aún cuando otros hilos no tengan peticiones por atender.

Los hilos también son útiles en la parte cliente. Un hilo podría estar generando resultados para ser utilizados por un servidor, mientras que un segundo es utilizado para transmitir estos datos al servidor. Un caso evidente de múltiples hilos en un cliente son los buscadores de páginas Web. Es común que un usuario desee acceder a más de una página a la vez, por lo que el buscador de páginas Web tendrá que manejar múltiples solicitudes de páginas.

En la programación concurrente, al crear un proceso multihilos se requiere de un gran cuidado. La principal dificultad es la compartición de objetos y las técnicas usadas para la coordinación y cooperación de hilos. Por ejemplo, en la figura 1.7.c, las peticiones de cola pueden perderse o duplicarse a menos que las manipulaciones del puntero del hilo sean cidadosamente coordinadas. Convencionalmente el lenguaje más utilizado para hacer uso de las características de los hilos es el lenguaje C, pero existen otros como Ada95, Modula-3 y más recientemente JAVA.

1.2.4 Concurrencia

La presencia de múltiples usuarios en un sistema distribuido es una fuente de peticiones concurrentes a sus recursos. Para atender múltiples solicitudes cliente se hace uso de la programación concurrente[22], en donde es indispensable la creación de procesos e hilos.

Los procesos son concurrentes si existen simultáneamente. Estos pueden funcionar en forma totalmente independiente unos de otros o pueden acceder a un mismo recurso, lo cual significa que en ocasiones requieren cierta sincronización y cooperación. Para ejemplificar este último caso, consideremos que existen muchos procesos concurrentes que leen y escriben objetos simultáneamente y no tienen problemas de ejecución; sin embargo, los resultados que arroja cada uno de ellos son imprecisos. Pudiera ser que mientras un proceso estaba leyendo un objeto, otro proceso estaba rescribiendo su valor o cuando un proceso estaba escribiendo objetos, tal vez otro proceso también estaba escribiendo sobre ellos. Es por ello, que la ejecución de procesos concurrentes requiere sincronizar sus operaciones.

[22] Un programa concurrente genera procesos que simulan ejecutarse simultáneamente.

1.2.4.1 Transacciones

Las técnicas para la administración descentralizada de datos fueron desarrolladas originalmente en el marco de los sistemas de bases de datos. Estos sistemas introdujeron las nociones de transacciones y operaciones atómicas, que son útiles para la administración de datos y control de concurrencia.

Una transacción es una secuencia de operaciones que forma un solo paso, transformando el servidor de datos de un estado consistente a otro y está destinada a ser atómica. En algunas ocasiones, los clientes requieren que las peticiones atendidas por el servidor se lleven a cabo de manera separada, es decir, que sean libres de las interferencias por operaciones ejecutadas concurrentemente por otros clientes y que todas las operaciones se completen exitosamente o que no tengan ningún efecto en servidores interrumpidos. Además de esta propiedad de atomicidad existen otras que [HaRe83] mediante el mnemónico 'ACID' , las menciona:

a) **Atomicidad.** Una transacción debe ser todo o nada. Una transacción se completa exitosamente y el efecto de todas sus operaciones son registradas en los objetos, o si fracasa (si falla o es abortada deliberadamente) no tiene efecto en ellos.
b) **Consistencia.** Una transacción, después de su ejecución, debe mantener el estado consistente del sistema.
c) **Aislamiento.** Cada transacción debe ser ejecutada sin interferir con otra, es decir, los efectos intermedios de una transacción no deben ser visibles a otras transacciones.
d) **Durabilidad.** Después de que se ha completado exitosamente una transacción todos sus efectos son salvados en un almacenamiento permanente[23].

Un servidor que ejecuta transacciones debe sincronizar sus operaciones para asegurar el aislamiento entre ellas. Una forma de hacerlo es ejecutar las transacciones de manera serial (una a la vez en un orden arbitrario). Desafortunadamente, esta solución no sería aceptable por servidores cuyos recursos son compartidos por múltiples usuarios. Para maximizar la concurrencia, cualquier servidor (soportando transacciones) debe asegurar que la ejecución de las transacciones tenga el mismo efecto que si se ejecutarán de forma serial, es decir, que sean serialmente equivalentes.

Antes de garantizar la equivalencia serial entre transacciones es importante saber que problemas existen al ser ejecutadas concurrentemente. Hay dos problemas principales en este tema: la pérdida de actualización y la recuperación inconsistente.

En la figura 1.8.a, podemos apreciar el problema de la pérdida de actualización en las operaciones que ejecutan las transacciones T y U, sobre las cuentas bancarias A, B y C. El saldo inicial de A es de $100, el de B es de $200 y el de C es de $300. La transacción T transfiere la cantidad de $20 de A a B (figura 1.8.a, operaciones 1, 4 y 5). Asimismo, la transacción U transfiere la cantidad de $20 de C a B (figura 1.8.a, operaciones 2, 3 y 6). El efecto total sobre la cuenta B después de ejecutar las transacciones T y U debería ser $240. Sin embargo, como las dos transacciones leen de B $200, el saldo final de B es de 220.

El problema de la pérdida de actualización ocurre cuando dos transacciones leen al mismo tiempo el valor de una variable y lo usan para calcular el nuevo valor. Esto no pasaría, si una transacción es ejecutada antes que otra, así la segunda transacción leería el valor escrito por la anterior.

[23] Se usa el término *almacenamiento permanente* para referirse a archivos en el disco u otro medio permanente.

Entonces, la transacción de lectura y la transacción de actualización se intercalan para obtener una equivalencia serial, como se muestra en la figura 1.8.b.

Op.	Transacción T		Transacción U	
1	saldo=b.lee	$200		
2			saldo = b.lee	$200
3			b.escribe(saldo+20)	$220
4	b.escribe(saldo+20)	$220		
5	a.retira(20)	$80		
6			c.retira(20)	$280

(a)

Op.	Transacción T		Transacción U	
1	saldo = b.lee	$200		
2	b.escribe(saldo+20)	$220		
3			saldo = b.lee	$220
4			b.escribe(saldo+20)	$240
5	a.retira(20)	$80		
6			c.retira(20)	$280

(b)

Figura 1.8. (a) Problema de pérdida de actualización. (b) Equivalencia serial.

Ahora se considera el problema de recuperación inconsistente relacionado con la ejecución de dos transacciones bancarias. En la figura 1.9.a, los saldos de las cuentas bancarias A y B son inicialmente de $200. La transacción T transfiere la suma de $100 de A a B (figura 1.9.a, operaciones 1 y 5), mientras que la transacción U obtiene la suma de los saldos de todas las cuentas en el banco (figura 1.9.a, operaciones 2, 3 y 4). Esta suma incluye las cuentas de A y B igual a $300, la cual, no es correcta porque T sólo ha ejecutado la parte del retiro de la transferencia a B. El problema de la recuperación inconsistente ocurre cuando una transacción de recuperación y una de actualización son ejecutadas concurrentemente. Para prever esta inconsistencia es conveniente realizar la intercalación de equivalencia serial de la transacción de recuperación y la transacción de actualización, como se muestra en la figura 1.9. b.

Op.	Transacción T		Transacción U	
1	a.retira(100)	$100		
2			total= a.lee()	$100
3			total=total+b.lee()	$300
4			total=total+c.lee()	$600
5	b.deposita(100)	$300		
6			...	

(a)

Op.	Transacción T		Transacción U	
1	a.retira(100)	$100		
2	b.deposita(100)	$300		
3			total= a.lee()	$100
4			total=total+b.lee()	$400
5			total=total+c.lee()	$700
6			...	

(b)

Figura 1.9. (a) Problema de recuperación inconsistente. (b) Equivalencia serial.

En los dos casos anteriores se mostró que el uso de la equivalencia serial, como un criterio para la ejecución de concurrencia correcta, previene la pérdida de actualización y la recuperación inconsistente. Sin embargo, para llevarla a cabo se debe determinar cual será el orden de ejecución de las operaciones en conflicto. Para ello, se tiene como base las operaciones de lectura y escritura (la operación de lectura accede al valor de un objeto y la operación de escritura cambia su valor), y las reglas de conflicto para un par de operaciones (ver la figura 1.10).

Operación de transacciones diferentes		Conflicto	Razón
Lectura	Lectura	No	El efecto de un par de operaciones de lectura no dependen del orden en que se ejecutan.
Lectura	Escritura	Si	El efecto de las operaciones de lectura y escritura depende del orden de su ejecución.
Escritura	Escritura	Si	El efecto de un par de operaciones de escritura depende del orden de su ejecución.

Figura 1.10. Reglas de conflicto para un par de operaciones.

Para cualquier par de transacciones, es posible determinar el orden de pares de operaciones en conflicto sobre un objeto y la equivalencia serial puede ser definida en los siguientes términos:

"Para que dos transacciones sean serialmente equivalentes, es necesario y suficiente que todos los pares de operaciones en conflicto de las dos transacciones sean ejecutadas en el mismo orden sobre el objeto que ellas acceden."

Op.	Transacción T	Transacción U
1	x = lee(i)	
2	escribe(i,10)	
3		y = lee(j)
4		escribe(j,30)
5	escribe(j,20)	
6		z = lee(i)

Figura 1.11. Intercalación de operaciones de las transacciones T y U.

Ahora se considera el ejemplo de la figura 1.11, donde se definen las transacciones T y U de la siguiente manera:

T : x = lee(i); escribe(i,10); escribe(j,20)
U: y = lee(j); escribe(j,30); z = lee(i)

Cada acceso de las transacciones a los objetos *i* y *j* es serializada con respecto a la otra, porque T hace todos sus accesos a *i* antes que U lo haga (figura 1.11, operaciones 1 y 2) y U hace todos sus acceso a *j* después de que T lo hace (figura 1.11, operaciones 3 y 4). Sin embargo, el orden no es equivalencia serial porque el par de operaciones en conflicto no son hechas en el mismo orden en ambos objetos (figura 1.11, transacción 5 y 6). El orden de la equivalencia serial requiere una de las siguientes condiciones:

1. T accesa i antes que U y T accesa j antes que U.
2. U accesa i antes que T y U accesa j antes que T.

Entre los mecanismos más comunes que hacen uso de la equivalencia serial para el control de concurrencia están: el bloqueo, el control de concurrencia optimista y las etiquetas de tiempo.
Otro problema que existe en la ejecución de las transacciones son los abortos. La propiedad de atomicidad establece que se deben registrar todos los efectos de todas las transacciones ejecutadas y suprimir todos los efectos de las transacciones abortadas. Para éste último caso, se debe prevenir el efecto de la transacción abortada con respecto a otras ejecutadas concurrentemente.
Una consecuencia del aborto de transacciones son las llamadas *lecturas pobres* y *escrituras prematuras*. El problema de *lectura pobre* es causada por la interacción entre una operación de lectura en una transacción y una operación de escritura temprana en otra transacción sobre el mismo objeto. En la figura 1.12.a se muestra este caso, la transacción T obtiene el saldo de A y le suma $10 más, posteriormente U obtiene el saldo de la cuenta de A y le establece $20 más. Las dos transacciones se han ejecutado serialmente. Sin embargo, T aborta después de que U se ha ejecutado y ésta utiliza un valor que nunca existió, entonces A no tiene un saldo correcto.
El problema de la escritura prematura está relacionada con las operaciones de escritura sobre el mismo objeto entre transacciones diferentes. En la figura 1.12.b, la transacciones T y U escriben

en la cuenta A . El saldo inicial de A es de $100 . Si T y U, abortan el saldo de A debería ser de $100. Pero como U aborta después que T, U regresa el saldo de A a su estado anterior que es de $105 y no de $100 como debería de ser.

No	Transacción T	Transacción U
1	saldo=a.lee() $100	
2	a.escribe(saldo+10) $110	
3		saldo=a.lee() $110
4		a.escribe(saldo+20) $130
5		transacción realizada
6	transacción abortada	

(a)

No	Transacción T	Transacción U
1	a.escribe(105) $105	
2		a.escribe(110) $110
3	transacción abortada	
4		transacción abortada

(b)

Figura 1.12. (a) Lectura pobre. (b) Escritura prematura.

Generalmente, se requiere que las transacciones sean retardadas en sus operaciones de lectura y escritura para evitar lecturas pobres y escrituras prematuras. Las ejecuciones de transacciones son llamadas estrictas si el servicio retarda ambas operaciones de lectura y escritura sobre un objeto hasta que todas las transacciones que escribieron anteriormente en el objeto hallan completado su ejecución o abortado.
A continuación, se describen los mecanismos de control de seguridad anteriormente mencionados y posteriormente, se realiza una comparación de ellos.

Bloqueo

Entre los diferentes mecanismos para lograr la equivalencia serial está el *bloqueo exclusivo*. En este esquema, el servidor intenta bloquear el objeto que va a ser usado por cualquier operación de una transacción cliente. La figura 1.13, muestra el uso de bloqueos exclusivos y antes que las transacciones sean ejecutadas, las cuentas A, B y C no están bloqueadas. Cuando la transacción T está por leer la cuenta B, el servidor la bloquea para T (figura 1.13, operación 1). Pero cuando la transacción U está por leer B, ésta debe esperar porque se encuentra bloqueada por T (figura 1.13, operación 3). Al terminar de ejecutarse T, B es desbloqueada por lo que la transacción U es reanudada (figura 1.13, operación 5). Entonces, cuando un cliente solicita acceder al objeto que está bloqueado, la petición es suspendida y el cliente debe esperar hasta que el objeto sea desbloqueado.

Op.	Transacción T		Transacción U	
	Operaciones	Bloqueos	Operaciones	Bloqueos
1	abre-transacción saldo:=b.lee( )	bloquea B		
2	b.escribe(saldo+20)			
3			abre-transacción saldo := b.lee( )	espera que T desbloquee a B
4	saldo:=a.retira(20) cierra-transacción	bloquea A desbloquea A y B		
5			b.escribe(20)	bloquea B
6			c.retira(20) cierrar-transacción	bloquea C desbloquea B y C

Figura 1.13. Bloqueos exclusivos.

La equivalencia serial requiere que todas las operaciones realizadas sobre los objetos por las transacción sean serializadas con respecto a las operaciones de otras transacciones, es decir, si las operaciones de cada una de las transacciones se encuentran en conflicto entre sí, entonces deben ser ejecutadas en el mismo orden. Para asegurar esta, una transacción no se le permite hacer nuevos bloqueos hasta que haya liberado un bloqueo. La primera fase de cada transacción es llamada *fase en crecimiento*, durante la cual bloqueos nuevos son adquiridos. En la segunda fase, los bloqueos son liberados *fase de reducción*. Esto es conocido como el ***bloqueo de dos fases***.

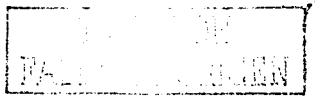
Bajo un esquema estricto de ejecución, cualquier bloqueo, aplicado durante la ejecución de una transacción es detenido hasta que la transacción sea ejecutada o abortada. A esto se llama el ***bloqueo estricto de dos fases***. La presencia de los bloqueos previenen otras transacciones de lectura y de escritura sobre objetos, y cuando una transacción se ejecuta, para asegurar el restablecimiento, los bloqueos son retenidos hasta que todos los objetos han sido escritos o almacenados permanentemente. Una desventaja de este tipo de bloqueo es que reduce la concurrencia más de lo necesario. Para ello es preferible adoptar otro esquema de bloqueo que consiste en controlar el acceso a cada unidad de datos para que diferentes transacciones concurrentes puedan realizar la acción de lectura y sólo una transacción realice la de escritura, pero no ambas al mismo tiempo. Este esquema es comúnmente conocido como ***muchos lectores/un escritor***, y utiliza dos tipos de bloqueo que son: el *bloqueo de lectura* y el *bloqueo de escritura*. Antes de que una operación de lectura de una transacción sea ejecutada, el servidor intenta establecer un bloqueo de lectura sobre un objeto. Y antes de que una operación de escritura sea ejecutada, el servidor intenta establecer un bloqueo de escritura sobre un objeto. Sin embargo, un servidor no está habilitado para establecer un bloqueo inmediatamente, lo que hace es mantener la transacción esperando hasta que la pueda realizar, por lo que nunca rechaza las peticiones de los clientes. Todas las transacciones que leen el mismo objeto comparten el bloqueo de lectura, también conocido como *bloqueos compartidos*. Las reglas entre un par de operaciones en conflicto dicen que:

1. Si una transacción ha ejecutado una operación de lectura sobre un objeto en particular, entonces una transacción concurrente U no debe escribir en el objeto hasta que T sea completamente ejecutado o abortada.
2. Si una transacción ha ejecutado una operación de escritura sobre un objeto en particular, entonces una transacción concurrente U no debe leer o escribir en el objeto hasta que T sea completamente ejecutado o abortada.

En la figura 1.14, se muestra la compatibilidad de los bloqueos de lectura y escritura sobre un objeto en particular. Si un objeto tiene un bloqueo ya establecido (columna izquierda), entonces otro bloqueo que llegue sobre el mismo objeto estará permitido o deberá esperar, dependiendo de la operación de lectura o escritura que desea realizar (columna derecha).

Para un objeto	**Bloqueo solicitado**	
Bloqueo ya establecido	**Lectura**	**Escritura**
Ninguno	Permitido	Permitido
Lectura	Permitido	Espera
Escritura	Espera	Espera

Figura 1.14. Compatibilidad de los bloqueos de lectura y escritura sobre un objeto.



Los dos problemas más comunes entre transacciones que operan sobre los mismos objeto son las *recuperaciones inconsistentes y* las *pérdidas de actualización*.

Las recuperaciones inconsistentes son prevenidas mediante la ejecución de una transacción de lectura antes o después de una transacción de actualización. Si la recuperación es primero, sus bloqueos de lectura retardan la transacción de actualización. Si es después, su petición para el bloqueo de lectura causa que ésta sea retardada hasta que la transacción de actualización se haya completado.

Con respecto a la pérdida de actualización, ésta ocurre cuando dos transacciones leen un valor de un objeto y lo usan para calcular otro nuevo valor. Para prevenirlo, las transacciones posteriores deben retardar sus lecturas hasta que las anteriores se hallan completando. Para ello, cada transacción debe establecer un bloqueo de lectura sobre el objeto que desea leer y cuando requiera escribir en él, el bloqueo será promovido a uno de escritura, de tal forma que, cuando una transacción subsecuente solicite el bloqueo de lectura de ese objeto, será retardada hasta que la transacción actual haya finalizado. El término *promover un bloqueo* se refiere a la conversión de un bloqueo a un bloqueo fuerte, que es un bloqueo más que exclusivo. Sin embargo, no siempre es posible realizarlo. Una transacción con un bloqueo de lectura, que es compartido con otras transacciones, no puede promoverlo inmediatamente a uno de escritura porque surgirían conflictos con los bloqueos de lectura retenidos por otras transacciones. Por lo tanto, una transacción que desea realizar un bloqueo de escritura debe esperar la liberación de los bloqueos de lectura. La tabla de compatibilidad de bloqueos de la figura 1.14, muestra cuales bloqueos son exclusivos. El bloqueo de escritura no permite otros bloqueos de lectura ni de escritura. En cambio, el bloqueo de lectura sólo permite bloqueos de lectura. Esto conduce a que un bloqueo de escritura es más exclusivo que un bloqueo de lectura.

Las reglas de uso de bloqueos en una implementación de *bloqueo estricto de dos fases* son resumidas en la figura 1.15. El bloqueo es realizado por el servidor cuando las operaciones de lectura y escritura son solicitadas, y desbloqueadas al realizar las operaciones ejecutar o abortar.

1. Cuando una operación accede a un objeto en una transacción:
   a) Si el objeto no esta bloqueado, es bloqueado y la operación continúa.
   b) Si el objeto tiene un conflicto de bloqueo establecido por otra transacción, la transacción debe esperar hasta que sea desbloqueada.
   c) Si el objeto no tiene conflictos de bloqueo con otras transacciones, el bloqueo es compartido y la operación continúa.
   d) Si el objeto ya ha sido bloqueado en la misma transacción, el bloqueo será promovido en caso de que sea necesario y la operación continúa. (Cuando el bloqueo no es promovido, la regla (b) es usada).
2. Cuando una transacción es ejecutada o abortada, el servidor desbloquea todos los objetos que bloqueo durante la transacción

Figura 1.15. Las reglas para el uso de bloqueos.

Bloqueos Mutuos. El uso de bloqueos puede llevar a bloqueos mutuos. En la figura 1.16, las transacciones T y U desean escribir en B y A respectivamente, pero cada una de ellas debe esperar hasta que sean desbloqueadas. Esta situación es un bloqueo mutuo, por que T espera que U desbloquee a B y U espera que T desbloquee a B.

Op.	Transacción T		Transacción U	
	Operaciones	Bloqueos	Operaciones	Bloqueos
1	a.deposita(100)	bloqueo de escritura en A		
2			b.deposita(200)	bloqueo de escritura en B
3	b.deposita(100)	espera que U desbloquee B		
4	...		a.deposita(200))	espera que T desbloquee A
5			...	

Figura 1.16 Ejemplo de bloqueo mutuo

El bloqueo mutuo es un estado en el cual cada miembro de un grupo de transacciones está esperando que sea liberado el bloqueo.

Bloqueos con tiempo limitado. Los bloqueos con tiempo limitado es un método para la resolución de bloqueos mutuos. Cada bloqueo está sujeto a un periodo en el cual es invulnerable, después de ese tiempo, llega a ser vulnerable. Si existe una o más transacciones en espera de acceder un objeto protegido por un bloqueo vulnerable, éste es desbloqueado y las transacciones que están esperando reanudan sus actividades. La transacción cuyo bloqueo ha sido eliminado es abortada. Usando este mecanismo se puede resolver el bloqueo mutuo de la figura 1.15, en el cual, el bloqueo de escritura para T en A llega a ser vulnerable después de que acaba su tiempo límite. La transacción U esta esperando adquirir un bloqueo de escritura en A. Entonces, T es abortada y libera su bloqueo en A, permitiendo a U reiniciar y completar la transacción.
Una de las consecuencias del bloqueo con tiempo limitado, es el aborto de transacciones sin necesidad de hacerlo. Es probable que las transacciones que están esperando el bloqueo no provoquen ningún bloqueo mutuo con aquellas que tienen el bloqueo, sin embargo como se acaba su tiempo de invulnerabilidad son abortadas. Estos bloqueos también pueden afectar mucho a sistemas con gran actividad. Como el número de transacciones es grande, también lo será el número de transacciones con tiempo límite pero posiblemente serán penalizadas después de un tiempo. No es fácil decidir cuál es la longitud apropiada del periodo.

Incremento de concurrencia. En este esquema se permite que una transacción escriba versiones tentativas de los objetos mientras que otras transacciones leen las versiones ejecutadas de ese objeto. Las operaciones de lectura únicamente esperan, si otra transacción es ejecutada concurrentemente sobre el mismo objeto. Este mecanismo llamado *bloqueo versión dos*, permite más concurrencia que los *bloqueos lectura-escritura*, sin embargo, las transacciones de escritura se arriesgan a esperar o ser rechazadas cuando intentan ejecutarse. Y las transacciones que solicitan ejecutarse esperan hasta que las transacciones de lectura se hayan completado. El bloqueo mutuo puede ocurrir cuando las transacciones están esperando ejecutarse, en este caso, las transacciones son abortadas. Este mecanismo fue implementado en el servidor de archivos XDFS[24] [WXde], el cual usa tres tipos de bloqueos: un *bloqueo de lectura*, un *bloqueo de escritura* y un *bloqueo de ejecución*. Antes de que una operación de lectura de una transacción sea ejecutada, el servidor intenta establecer un bloqueo de lectura sobre un objeto, este resulta exitoso a menos que el objeto tenga un bloqueo de ejecución, en tal caso la transacción espera. Antes que una operación de escritura de una transacción es ejecutada, el servidor intenta establecer un bloqueo de escritura sobre un objeto, el intento para establecer un bloqueo de escritura es exitoso

[24] Del inglés *"Xdelta File System"*

a menos que el objeto tenga un bloqueo de escritura o un bloqueo de ejecución, en tal caso las transacciones esperan.

Bloqueo realizado	Bloqueo a realizar		
	Lectura	Escritura	Ejecución
Ninguno	OK	OK	OK
Lectura	OK	OK	Esperar
Escritura	OK	Esperar	-
Ejecución	Esperar	Esperar	-

Figura 1.17. Compatibilidad de los bloqueos de lectura, escritura y ejecución.

Cuando el servidor recibe una petición para ejecutar una transacción, intenta convertir todos los bloqueos de escritura de una transacción a bloqueos de ejecución. Si alguna unidad de datos tiene bloqueos de lectura pendientes, la transacción debe esperar hasta que el bloqueo sea liberado. La compatibilidad de los bloqueos de lectura, escritura y ejecución se muestra en la tabla de la figura 1.17.

Existen dos diferencias principales entre el bloqueo versión dos y el bloqueo lectura-escritura. Por una parte, las operaciones de lectura en el esquema de bloqueo versión dos son retardadas únicamente durante la ejecución de transacciones, en lugar de toda la ejecución de la transacción. Por otra, las operaciones de lectura de una transacción pueden causar retardo en ejecutar otras transacciones.

Control de concurrencia optimista

[KuRo81] identificaron algunas desventajas en el bloqueo. Observaron que las transacciones de lectura, las cuales no afectan la integridad de los datos, deben usar un bloqueo para garantizar que los datos leídos no son modificados por otras transacciones al mismo tiempo. Así también, para evitar abortar en cascada, los bloqueos no son liberados hasta el final de la transacción, como consecuencia, el nivel de concurrencia se reduce significantemente. Esto también ocurre en la prevención de los bloqueos mutuos. Debido a estas desventajas, [KuRo81] propusieron un enfoque optimista para la serialización de las transacciones y evitar las fallas del bloqueo. Este enfoque se basa en la baja probabilidad que tienen dos transacciones cliente de acceder el mismo objeto al mismo tiempo. Las transacciones son realizadas como si no existiera la posibilidad de un conflicto entre ellas y cuando completan su tarea emiten la petición *transacción-cerrada*. Sin embargo, cuando ocurre un conflicto, alguna transacción es abortada y necesitará ser reiniciada por el cliente. Cada transacción consiste de las siguientes tres fases:

1. **Fase trabajo**. Durante esta fase, cada transacción tiene una versión tentativa de los objetos que va actualizar. Esta es una copia de la versión ejecutada más reciente del objeto. El uso de versiones tentativas permiten a una transacción abortar (con ningún efecto sobre los objetos), ya sea durante la fase de trabajo o si falla la validación debido a conflictos con otras transacciones. Las operaciones de lectura son ejecutadas inmediatamente, si una versión tentativa para esa transacción ya existe, de otra forma se obtiene el valor del objeto escrito más reciente. Las operaciones de escritura registran los nuevos valores de los objetos como valores tentativos (los cuales son invisibles a otras transacciones). Cuando existen varias transacciones concurrentes, los diferentes valores tentativos de los objetos pueden coexistir.

Es importante observar que todas las operaciones de lectura son llevadas a cabo sobre versiones ejecutadas de un objeto, por lo que las lecturas pobres[25] no pueden ocurrir.

2. **Fase de validación**. Cuando la petición de *transacción-cerrada* es recibida, la transacción entra en la fase de validación, en donde se revisa que sus operaciones sobre el objeto no este en conflicto con operaciones de otras transacciones. Si la validación es exitosa, la transacción puede ejecutarse. Si la validación fracasa, mediante algún proceso se resuelve el conflicto o en el peor de los casos la transacción abortada.
3. **Fase de actualización.** Si una transacción es válida, todos los cambios registrados en sus versiones tentativas se hacen permanentes. Las operaciones de sólo lectura pueden ejecutarse inmediatamente al pasar la validación. Y las transacciones de escritura están listas para ejecutarse, una vez que las versiones tentativas de los objetos han sido registradas en un almacenamiento permanente.

La fase de validación usa las reglas de conflicto *lectura/escritura* para asegurar que las operaciones de una transacción son serialmente equivalentes con respecto a todas las transacciones con las que se traslapa. En la validación, cada transacción se le asigna un número entero (en forma ascendente), y si la transacción es válida y completada exitosamente conservará su número, pero si fracasa será abortada o si la transacción sólo fue de lectura, el número es liberado para volverlo a asignar. La prueba de validación se basada en los conflictos entre operaciones de dos transacciones Ti y Tj,
Y para que una transacción Tj sea serializable con respecto a una transacción Ti, sus operaciones deben seguir las reglas de la figura 1.18, donde i<j e i, j $\in$ Z.

Ti	Tj	Regla
Escritura	Lectura	1. Ti no debe leer objetos escritos por T
Lectura	Escritura	2. Tj no debe leer objetos escritos por Ti
Escritura	Escritura	3. Ti no debe escribir objetos escritos por Tj y Tj no debe escribir objetos escritos por Ti.

Figura 1.18. Conflictos entre operaciones de dos transacciones Ti y Tj.

Para prevenir el traslape de transacciones, las fases de validación y de actualización pueden ser implementadas como una sección crítica por lo que a esta sólo puede entrar un cliente.
Existen dos formas de validación: hacia atrás y hacia delante. La *validación hacia atrás* revisa las transacciones que están en la fase de validación con aquellas anteriores con las que se traslapa. La *validación hacia delante* revisa la transacción que está en la fase de validación con otras posteriores, las cuales están activas.

Validación hacia atrás. Como todas las operaciones de lectura de transacciones anteriores se ejecutaron antes de que empezara la validación de Tj, no son afectadas por la escritura de la transacción actual (se cumple con la regla 1). Sin embargo, la validación de la transacción Tj debe revisar que su lectura no se traslape con cualquiera de las escrituras establecidas por transacciones anteriores con las que se traslape (regla 2). Si existe cualquier traslape la validación fracasa.
En la figura 1.19, se muestra el traslape de transacciones que deben ser consideradas en la validación de una transacción Tj. El tiempo se incrementa de izquierda a derecha. La transacción

[25] Del inglés "*Dirty reads*"

T1 se ejecuta antes de que Tj empiece. T2 y T3 son ejecutadas antes de que Tj termine su fase de trabajo. En el caso de la validación hacia atrás, debido a que, T2 y T3 se ejecutan antes de que Tj termina su fase de trabajo, la lectura de Tj debe ser comparado con la escritura de T2 y T3. En caso de un conflicto, la única manera de resolverlo es abortar la transacción Tj que está siendo validada.

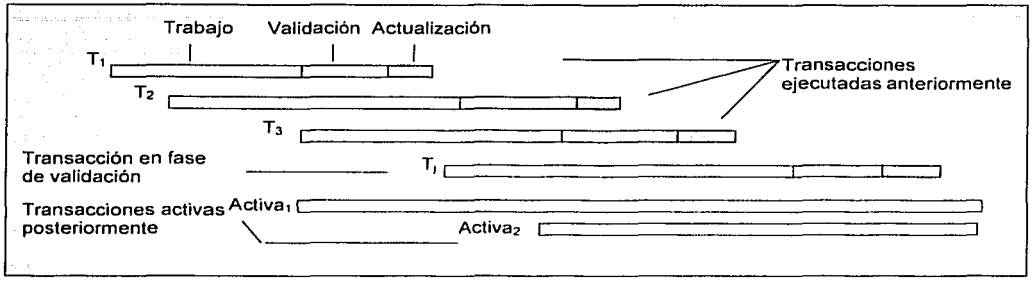


Figura 1.19. Validación de transacciones

Además, la validación hacia atrás requiere que las escrituras de versiones ejecutadas anteriormente de los objetos correspondientes a transacciones recientemente ejecutadas sean retenidas hasta que no existan transacciones no validadas traslapadas con las que haya conflicto. En un ambiente con muchas transacciones esto puede ser un problema. Por ejemplo, en la figura 1.19, las escrituras T1, T2, T3 y Tj deben ser retenidas hasta que la transacción activa1 se complete.

Validación hacia adelante. En la validación de la transacción Tj, la escritura es comparada con las lecturas de todas las transacciones activas traslapadas, aquellas que todavía están en su fase de trabajo (Regla 1). La regla2 se cumple automáticamente porque las transacciones activas no escriben hasta después de que Tj se ha completado.
En la figura 1.19, la escritura de la transacción Tj debe ser comparada con las lecturas de las transacciones activa1 y activa2. Debido a que la lectura de la transacción que está siendo validada no está incluida en la revisión, las transacciones de sólo lectura siempre pasan la fase de validación. Como las transacciones que están siendo comparadas con la transacción validada, aún están activas, se tiene la opción de abortar la transacción validada, abortar todas las transacciones activas en conflicto, retrasar la validación después de que las transacciones en conflicto hayan terminado o tomar otra alternativa para resolver el conflicto.

Comparación entre la validación hacia adelante y hacia atrás. La validación hacia adelante es más flexible en la resolución de conflictos, mientras que la validación hacia atrás únicamente permite abortar la transacción que está siendo validada. En general, las lecturas de las transacciones son más largas que las escrituras, por lo que en la validación hacia atrás las lecturas largas se comparan contra escrituras anteriores, mientras que en la validación hacia adelante se revisan pequeñas escrituras contra las lecturas de transacciones activas. Por lo tanto, la validación hacia atrás requiere de un almacenamiento para las escrituras anteriores, hasta que ya no sean requeridas. Por otra parte, la validación hacia adelante tiene que permitir que nuevas transacciones empiecen durante la fase de validación.

Ordenamiento de marcas de tiempo

En el control de concurrencia, basado en el ordenamiento de marcas de tiempo, cada operación de una transacción es validada cuando se lleva a cabo. Si la operación no es válida la transacción es abortada inmediatamente y el cliente puede volver a iniciarla. Las peticiones de las transacciones pueden ser ordenadas totalmente de acuerdo a sus marcas de tiempo. La regla de ordenamiento de las marcas de tiempo se basa en el conflicto entre las operaciones donde:

La operación de escritura de una transacción sobre un objeto es válida si ese objeto fue leído y escrito por última vez por transacciones anteriores. La operación de lectura de una transacción sobre un objeto es válida únicamente si el objeto fue escrito por última vez por una transacción anterior.

Esta regla asume que existe únicamente una versión de cada objeto y restringe el acceso a una transacción a la vez. Si cada transacción tiene su propia versión tentativa de cada objeto, entonces múltiples transacciones concurrentes pueden acceder al objeto. Para que la regla de ordenamiento también asegure que las versiones tentativas de cada objeto sean ejecutadas en el orden determinado por las marcas de tiempo de las transacciones que las hicieron, las transacciones siguientes deberán esperar lo necesario para que las transacciones anteriores completen sus escrituras. Las operaciones de escritura serán ejecutadas después de la operación *cerrar-transacción*, sin hacer esperar al cliente. Pero el cliente deberá esperar cuando las operaciones de lectura no son ejecutadas debido a que las transacciones anteriores todavía no terminan Esta medida no permite la existencia de bloqueos mutuos.
A continuación se describirá la forma de control de concurrencia basado en las marcas de tiempo y adoptado por el sistema SDD-1 y descrito por [CePe85].
Las operaciones de escritura son registradas en versiones tentativas de los objetos y son invisibles a otras transacciones hasta que una petición *cerrar-transacción* es emitida y, por lo tanto la transacción es completada. Cada objeto tiene una marca de tiempo de escritura y un conjunto de versiones tentativas, cada una de las cuales tienen una marca de tiempo asociada y un conjunto de marcas de tiempo de lectura. La marca de tiempo de escritura del objeto es menor que cualquiera de sus versiones tentativas y el conjunto de marcas de tiempo de lectura pueden ser representadas por su miembro máximo. Siempre que una operación de escritura sobre un objeto es aceptada, el servidor crea una nueva versión tentativa del objeto con un conjunto de marcas de tiempo de escritura para la marca de tiempo de la transacción. Una operación de lectura de una transacción es dirigida a la marca de tiempo de escritura máxima menor que la marca de tiempo de la transacción. Cuando una operación de lectura de la transacción sobre una unidad de datos es aceptada, el servidor agrega las marcas de tiempo de la transacción a su conjunto de marcas de tiempo de lectura. Cuando una transacción es ejecutada, el valor de las versiones tentativas llegan a ser los valores de las unidades de datos y las marcas de tiempo de las versiones tentativas llegan a ser las marcas de tiempo de las correspondientes unidades de datos.
En el ordenamiento de la marca de tiempo, el servidor revisa si cada petición de una transacción para una operación de lectura o escritura sobre una unidad de datos conforma las reglas de conflicto (ver figura 1.20). Una petición hecha por la actual transacción Tj puede ocasionar conflictos con otras operaciones hechas por otras transacciones, Ti cuyas marcas de tiempo

indican que deben ser después que Tj. Estas reglas son mostradas en la siguiente tabla, en la cual Ti>Tj significa que ti es después que Tj y Ti > Tj significa que Tj es antes que Tj.

Ti	Tj	Regla
Escritura	Lectura	1. Tj no debe de escribir sobre una unidad de datos que ha sido leída por cualquier Ti donde Ti>Tj esto requiere que Tj >= a la máxima marca de tiempo de lectura de una unidad de datos.
Escritura	Escritura	2. Tj no debe escribir una unidad de datos que ha sido escrita por cualquier Ti donde Ti > Tj esto requiere que Tj> al máxima marca de tiempo de escritura de la unidad de dato ejecutada.
Lectura	Escritura	3. Tj no debe leer una unidad de datos que ha sido escrita por cualquier Ti cuando Ti>Tj esto implica que Tj no puede leer si Tj < marca de tiempo de escritura de la versión ejecutada de la unidad de datos.

Figura 1.20. Reglas de conflicto.

Regla de escritura en el ordenamiento de la marca de tiempo. Combinando las reglas 1 y 2, tenemos la siguiente regla para decidir si aceptar una solicitud de operación de escritura por una transacción Tj sobre un objeto D:

```
If (Tj >= a la máxima marca de tiempo de lectura sobre D &&
  Tj > a la marca de tiempo de la versión ejecutada de D )
    Ejecuta la operación de escritura sobre la versión tentativa de D con la marca de tiempo de escritura Tj
else /*La escritura es demasiado tarde*/
    Aborta la transacción Tj
```

La figura 1.21 ilustra la acción de una operación de escritura de la transacción T3. En los casos (a), (b) y (c), T3 > marca de tiempo de escritura sobre la versión ejecutada del objeto y la versión tentativa con la marca de tiempo de escritura de T3 es insertada en el lugar apropiado en la lista de versiones tentativas ordenadas mediante las marcas de tiempos de sus transacción. En el caso (d), T3< la marca de tiempo de escritura sobre la versión ejecutada del objeto y la transacción es abortada.

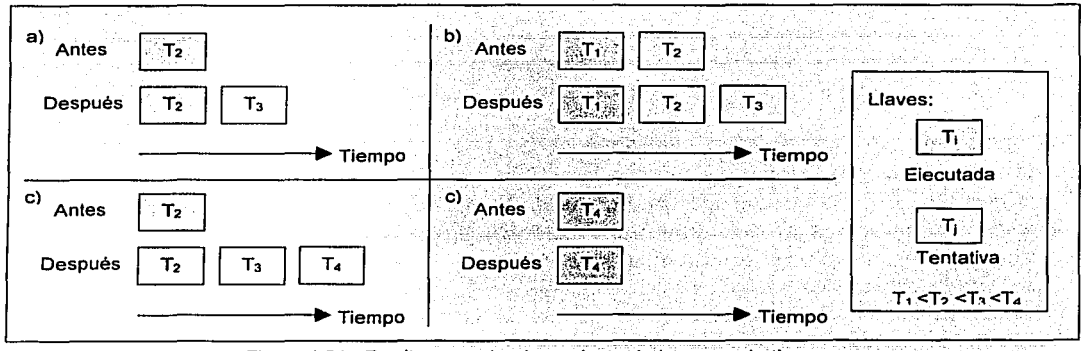


Figura 1.21 . Escritura en el ordenamiento de la marca de tiempo.

Regla de lectura en el ordenamiento de la marca de tiempo. Usando la regla 3 tenemos la siguiente condición para decidir si aceptar, esperar o rechazar una operación de lectura solicitada por la transacción Tj sobre el objeto D:

```
If (Tj > marca de tiempo de escritura sobre la versión ejecutada de D ) {
   Dseleccionado es la versión de D con la máxima marca de tiempo de escritura <=Tj
   If (Dseleccionado es ejecutada)
      Realizar la operación de lectura sobre la versión Dseleccionado
   else
      Esperar hasta que la transacción que hizo la versión Dseleccionado se ejecute o aborte,
      entonces volver aplicar la regla de Lectura.
} else   Abortar la transacción Tj
```

La figura 1.22 ilustra la acción de una operación de lectura de la transacción T3. En cada caso, la versión cuya marca de tiempo es menor o igual a T3 es seleccionada. Si tal versión existe, es indicada con una línea. En los casos (a) y (b) la operación de lectura usa una versión ejecutada. En (c), la operación de lectura una versión tentativa y debe esperar hasta que la transacción anterior se ejecute o aborte. Por último, en (d), no existen versiones adecuadas para la lectura y la transacción T3 es abortada.

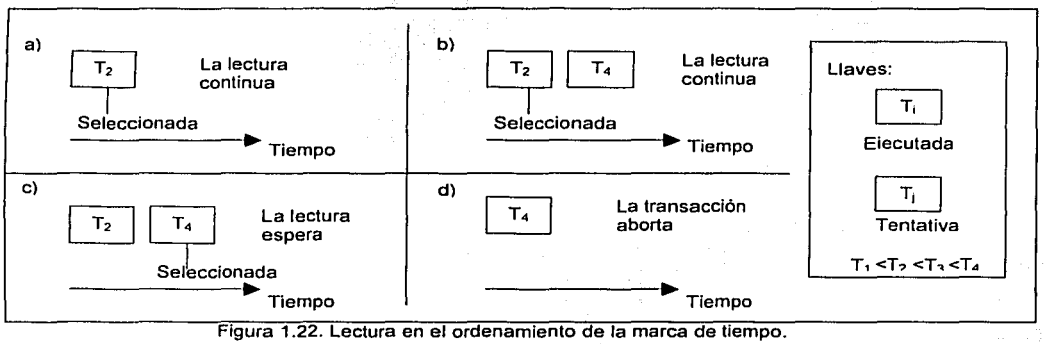


Figura 1.22. Lectura en el ordenamiento de la marca de tiempo.

La regla de lectura del ordenamiento de marca de tiempo retrasa una operación de lectura de una transacción sobre cualquier objeto hasta que todas las transacciones que habían escrito previamente en ese objeto se han ejecutado o abortado. La ejecución de versiones en orden asegura que la ejecución de una operación de escritura sobre un objeto sea retrasada hasta que todas las transacciones que escribieron previamente en ese objeto ya se hayan ejecutado o abortado.

1.2.4.2 Comparación de los métodos de control de la concurrencia

Anteriormente, se ha descrito tres métodos de control de concurrencia: el *bloqueo*, el *control de concurrencia* y el *ordenamiento de marcas de tiempo*. A continuación se realiza una comparación entre ellos, así como de sus diferentes versiones.

El método de *ordenamiento de marca de tiempo* es similar al *bloqueo de dos fases*, en ambos se usan enfoques pesimistas, en los cuales los conflictos entre transacciones son detectados cuando

cada objeto es accesado. Una diferencia entre ellos es el orden de serialización, mientras el *ordenamiento de marca de tiempo* utiliza el orden de serialización estática (cuando una transacción empieza), el *bloqueo de dos fases* lo hace dinámicamente (de acuerdo al orden en que los objetos son accesados). Otra diferencia se encuentra en la selección de la estrategia cuando es detectado un conflicto al acceder a un objeto. El *ordenamiento de marca de tiempo* aborta inmediatamente la transacción, mientras que el *bloqueo* hace que la transacción espere pero con la posibilidad de abortar para evitar el bloqueo mutuo.

En algunos trabajos que se han realizado, se ha observado que el *ordenamiento de marcas de tiempo* es mejor para transacciones donde las operaciones de lectura son predominantes y para transacciones donde existan más escritores que lectores, es preferible el *bloqueo*.

Con respecto al *método de control de concurrencia optimista*, todas las transacciones son permitidas pero algunas son abortadas cuando intentan ejecutarse, en la *validación hacia adelante* las transacciones son abortadas tempranamente. Esto resulta una operación eficiente cuando hay pocos conflictos pero en un sistema con mucha actividad, un número grande de transacciones tendría que ser reiniciado cuando una transacción es abortada.

Método	Tipo	Función	Ventaja	Desventaja
Bloqueo	Estricto de dos fases	Bloquea el objeto que esta siendo utilizado por la transacción hasta que esta termina o es abortada.	Previenen otras transacciones de lectura y de escritura sobre objetos.	Reduce la concurrencia más de lo necesario. Puede surgir el problema de bloqueo mutuo.
	Muchos lectores/un escritor	Controla el acceso a cada objeto para que diferentes transacciones concurrentes puedan realizar la acción de lectura y sólo una transacción realice la de escritura, pero no ambas al mismo tiempo.	Aumenta la concurrencia en comparación con el bloqueo de dos fases estricto y permite la promoción de bloqueos.	Puede surgir el problema de bloqueo mutuo.
	Con tiempo limitado.	Cada bloqueo está sujeto a un periodo en el cual es invulnerable.	Resuelve el problema del bloqueo mutuo que surge en el esquema muchos lectores/un escritor.	Puede originarse el aborto de transacciones siendo innecesario.
	Versión dos	Cada transacción escribe versiones tentativas de los objetos a acceder, mientras que otras transacciones leen las versiones ejecutadas de esos objetos.	Ofrece más concurrencia que el esquema "muchos lectores/un escritor".	Puede ocurrir el bloqueo mutuo. Este problema se resuelve abortando las transacciones en conflicto.
Control de concurrencia optimista	Validación hacia atrás	Revisa las transacciones que están en la fase de validación con aquellas anteriores con las que se traslapa.	Las transacciones son realizadas como si no existiera la posibilidad de un conflicto entre ellas. Esto es una ventaja para sistemas que almacenan objetos poco concurridos.	Cuando existe un conflicto únicamente permite el aborto de la transacción.
	Validación hacia adelante	Revisa la transacción que está en la fase de validación con posteriores que están activas. Es más flexible en la resolución de conflictos.		Las transacciones pueden ser abortadas tempranamente.
Ordenamiento de marcas de tiempo		Las peticiones de las transacciones son ordenadas de acuerdo a sus marcas de tiempo.	No permite la existencia de bloqueos mutuos.	Las transacciones son abortadas cuando llegan muy tarde.

Figura 1.23 Tabla comparativa de mecanismos de control de concurrencia.

Por lo tanto, el *ordenamiento de las marcas de tiempo* es mejor que el *bloqueo estricto de dos fases* para transacciones de única lectura. Éste último es mejor cuando predominan las actualizaciones en las transacciones. Para el caso de que existan pocos conflictos entre operaciones de transacciones concurrentes y una actividad no muy grande por parte del servidor, es conveniente el enfoque de *concurrencia optimista.*

En la tabla de la figura 1.23, se pueden observar los diferentes tipos de bloqueo. El *bloqueo de muchos lectores un escritor* establece una mayor concurrencia que el *bloqueo estricto de dos fases*, ya que permite que diferentes transacciones realicen la operación de lectura sobre un mismo objeto. Sin embargo, la ejecución de una operación de escritura sólo es realizada por una transacción a la vez. En este esquema pueden surgir bloqueos mutuos cuando un par de transacciones esperan que sean liberados los objetos que ambas están bloqueando. Para resolver este problema, se puede implementar el *bloqueo con tiempo limitado*, donde cada bloqueo después de un periodo de tiempo se vuelve vulnerable, permitiendo a otras transacciones acceder el mismo objeto. El problema de este esquema es determinar el tiempo de invulnerabilidad del bloqueo, ya que dependiendo de esto, se puede o no prevenir abortos innecesarios. El último tipo de bloqueo presentado es el *bloqueo versión dos*, el cual ofrece una mayor concurrencia que el *bloqueo de muchos lectores un escritor*, debido al manejo de versiones; sin embargo, también puede surgir el problema del bloqueo mutuo.

1.3 Resumen

Los sistema colaborativos, conocidos también como sistemas groupware, han surgido para asistir a grupos de personas en el desarrollo de una tarea en común. Estos sistemas han incorporado diferentes herramientas síncronas o asíncronas para brindar una mejor comunicación, compartición de recursos y coordinación, a los usuarios que suelen trabajar en diferentes espacios y tiempos. Sin embargo, el diseño de un sistema groupware no sólo se basa en investigaciones realizadas en el área del CSCW o HCI, sino también se requieren otros fundamentos que resuelvan problemas de heterogeneidad, seguridad, escalabilidad, fallas, concurrencia y transparencia, los cuales son temas de estudio de los sistemas distribuidos.

Como se pudo apreciar en este capítulo, en los sistemas distribuidos se pueden hallar diferentes arquitecturas, las cuales se diferencia dependiendo de las tareas que realicen cada uno de sus componentes. Actualmente, para implementar un sistema distribuido, en su arquitectura se está considerando desarrollar una capa de software que permite enmascarar la heterogeneidad. Esta capa conocida como mediador puede ser diseñada bajo el modelo de llamadas a procedimientos remotos, el modelo de invocación a métodos remotos o el modelo programación orientada a eventos. La selección de uno de ellos depende de las facilidades que proporcione para realizar tareas de comunicación, paso de mensajes, ejecución de mecanismos, interacción con otros sistemas, etc. En el desarrollo de este trabajo se ha experimentado con la invocación a métodos remotos, específicamente JAVA RMI, ya que además de ser portable sobre una gran variedad de plataformas, soporta una distribución de objetos y datos, la cual es una de las características más comunes en los sistemas colaborativos.

En la arquitectura de un sistema distribuido, es común que un componente sea prestador de un servicio (proceso servidor) y que en él surjan cuellos de botella. Para controlarlos, han surgido las arquitecturas de múltiples hilos, de las cuales podemos citar al modelo cliente-servidor, servidores

intermediarios de Web y de memoria temporal, par de procesos y servicios proporcionados por múltiples servidores. Aunque estas arquitecturas proporcionan una medida para incrementar la concurrencia, también pueden ser consecuencia de problemas de inconsistencia de información, ocasionados por conflictos de operaciones de escritura y lecturas entre transacciones. Una solución a este problema es la implementación de un mecanismo de control de concurrencia en transacciones. En este capítulo se describieron los mecanismos de bloqueo, control de concurrencia optimista y ordenamiento de marcas de tiempo. Sin embargo, para fines de este trabajo de tesis, sólo es de interés el mecanismo de bloqueo, debido a la forma en que maneja la asignación de recursos compartidos. En el capítulo 3 se retomará el mecanismo de bloqueo.

Una vez revisados algunos fundamentos esenciales de los sistemas distribuidos, ahora corresponde explorar el campo de los sistemas colaborativos, en especial el de los sistemas de edición colaborativa, motivo del presente trabajo de investigación.

Capítulo 2

Estado actual de los sistemas de trabajo colaborativo asistido por computadora

El CSCW es una disciplina científica que motiva y valida el trabajo en grupo con ayuda de las computadoras, siendo atractiva a aquellos interesados en el diseño del software y el comportamiento social y organizacional, incluyendo gente de negocios, científicos de la computación, psicólogos, investigadores de la comunicación, entre otros.

El concepto de CSCW fue manejado en 1960 por Doug Englebart [WDou], quien llevó a cabo experimentos de trabajo en grupo usando fondos militares de Estados Unidos. En ellos, incluyó un sistema que pudo mezclar video y texto, implementando "multimedia compartida". El sistema que desarrolló fue síncrono, por lo que en cada sesión se enviaba una réplica tan pronto como fuera posible. Posteriormente, en 1970, Englebart se enfocó en la aplicación de CSCW para la educación. Consistía en ayudar a pequeños grupo de maestros, estudiantes e investigadores, quienes deseaban comunicarse en tiempos diferentes (sistema asíncrono). En esa época, el sistema operativo UNIX, llegó a ser muy popular entre organizaciones educacionales debido a que permitía la comunicación entre computadoras y a su bajo costo.

Durante 1980, la introducción de la computadora personal de IBM causó una revolución en los negocios. Sin embargo, debido a que las computadoras personales fueron diseñadas para trabajar sin conexión con otras máquinas, no fueron de gran utilidad para el trabajo en equipo. El motivo principal para conectar la computadora personal a una red fue (y en la actualidad sigue siendo) la necesidad de compartir información y recursos, pero esto también condujo a pensar en cómo resolver los problemas que se pudiesen suscitar (por ejemplo, fallas de comunicación, inconsistencia de datos[26], cuellos de botella, etc.). Para finales de los años 80, las redes de computadoras empezaron a surgir, entre ellas Novell [WNov]. Esto dio un mayor impulso al desarrollo de los sistemas groupware.

En la actualidad, existen sistemas groupware con diferentes funcionalidades que sirven de apoyo al trabajo colaborativo. Algunos proponen aplicaciones para generar ideas, estructurarlas y finalmente evaluarlas. Otras herramientas, como los calendarios compartidos, asisten al manejo de citas, tareas, programación de juntas, etc. Incluso, existen otras que permiten trabajar sobre un recurso como los editores compartidos, las hojas de cálculo compartidas o el dibujo compartido. La meta de CSCW es descubrir formas de usar la tecnología para los grupos de trabajo a través del tiempo y el espacio. Por ello, las aplicaciones groupware no tratan de remplazar a la gente en una situación interactiva, en lugar de ello, es una herramienta mejorada para los procesos colaborativos.

[26] Varios usuarios pueden compartir un mismo recurso, por ejemplo un archivo. Si éste es modificado por todos los usuarios a la vez, cada quien creará una versión del mismo. Por lo que puede ocasionar que las aportaciones de cada usuario no persistan.

2.1 Criterios y requerimientos en el diseño de sistemas groupware

En el diseño de los sistemas groupware, existen criterios [JaLo92] y [Vol93]. Según [ReSe94], estos criterios pueden ser utilizados como requerimientos para la definición de un sistema groupware. A continuación se describen dichos criterios.

Espacio de trabajo. Es una simulación de un espacio físico mediante computadoras, en donde las personas pueden ver y manipular los objetos relacionados con las actividades que realizan. Los sistemas groupware que no muestran las acciones de otros usuarios son llamados *de colaboración transparente* [ReSe94]. Uno de los beneficios en el desarrollo del trabajo colaborativo es que cada usuario utiliza lo que hay en su espacio de trabajo individual, sin que sea modificado por otros.
Sin embargo, algunas veces es deseable que las modificaciones que se realizan en el sistema sean visibles para todos los usuarios involucrados. Tales sistemas con espacio de trabajo compartido son llamados *de colaboración conciente* [ReSe94] y son desarrollados específicamente como aplicaciones multiusuarios para el trabajo colaborativo. Otros elementos que deben contemplarse en los espacios de trabajo son los objetos de colaboración, las herramientas que pueden utilizarse en forma conjunta, la información sobre los usuarios y los eventos ocurridos en el espacio [Hen01].

Conciencia de grupo. Para lograr una colaboración consciente entre usuarios, se requiere mantener la compartición de la información y el conocimiento de las actividades grupales e individuales que se lleven a cabo, es decir, mantener una conciencia de grupo en los espacios de trabajo compartido. Esta es adquirida mediante elementos relacionados con diversas modalidades de conciencia, las cuales determinan la información que debe fluir entre los participantes:

- Quién interactúa con quién y qué están haciendo (Conciencia Informal).
- Qué conoce una persona de la otra en un contexto conversacional (Conciencia Social).
- Los roles, las responsabilidades y la posición de una persona frente a una tarea grupal (Conciencia de Grupo Estructural).
- La razón por la cual trabajan y comparten el conocimiento (Conciencia Organizacional)
- El propósito de una tarea, los requerimientos y las metas específicas del grupo (Conciencia de Tarea).

Además de las anteriores modalidades, [DoBe92], [Gut96] y [Mit95] establecen dos criterios claves para implementar la conciencia de grupo en los espacios compartidos: la información que debe capturarse acerca de la interacción de los participantes en el espacio y como debe ser presentada dicha información.
Existen otros elementos para fortalecer la conciencia en grupo propuestos, los cuales se asocian a preguntas relevantes que cada participante posiblemente se hace cuando trabaja en grupo [Gut96].

Visualización. En un espacio compartido es necesario reflejar un estado coherente del mismo. Por ejemplo, cuando se manipulan objetos remotos, los usuarios deben percibirlos como si fueran objetos locales para cada uno. Para la coherencia de la interfaz, se ha definido el concepto de vistas de espacio, las cuales se clasifican en cuatro tipos:

- WYSIWIS ("*What You See Is What I See*") [Ste87], en donde la vista del contenido del espacio es la misma para cada usuario.
- WYSIWYG ("*What You See Is What You Get*") [App87], en donde la vista representa exactamente cada objeto, tal y como existe dentro del espacio.
- WYSIWIMS ("*What You See Is What I May See*") [NePe91], permite perspectivas de vistas independientes y espacios de trabajo privados.
- WYSIMID ("*What You See Is What I Do*") [Gut96], en donde cada usuario visualiza, en forma síncrona o asíncrona, las acciones realizadas por su demás colegas.

El WYSIWIS está fuertemente relacionado a sistemas CSCW de colaboración transparente, ya que proporcionan información detallada sobre las ubicaciones y acciones actuales de los colaboradores. Los demás paradigmas se consideran como niveles del paradigma WYSIWIS.

Interacción. En comparación con las aplicaciones monousuario, existen nuevos requerimientos para la interacción en un ambiente groupware. La interacción puede ser *síncrona* o *asíncrona* dependiendo del tiempo de respuesta requerido. Por ejemplo, la edición conjunta de varios autores en la misma posición del texto se relaciona con el trabajo síncrono, mientras que enviar un correo electrónico es una tarea asíncrona. Otra clasificación en este criterio es la interacción *implícita* y *explícita*. La primera está relacionada con el objeto de trabajo, por ejemplo el compartir una imagen o texto establece una relación de trabajo con los usuarios. La segunda esta relacionada con la comunicación directa entre colaboradores mediante gestos, voz o transferencia de video. Finalmente, la interacción puede ser *formal* si los actos de los usuarios dependen de una posición y de roles, o *informal* como en una sesión colaborativa de compañeros iguales (sin jerarquías).
El intercambio de la información que fluye dentro del espacio se logra con técnicas de difusión de mensajes para la comunicación de grupos.

Coordinación. Para poder comunicarse y colaborar con el resto de la organización es necesario un medio que permita la integración y la adaptación armoniosa de todos los elementos que participan en un proceso. En la interacción, la coordinación depende del tamaño del grupo y de la individualidad de los miembros del grupo con respecto al modo en que ellos prefieren interactuar. Los grupos pequeños usualmente necesitan menos coordinación que los grupos grandes, por lo que, el modo de interacción cambia con el tamaño y la tarea del grupo. Por ejemplo, en una sesión de lluvia de ideas, cada miembro del grupo se comunica espontáneamente, en contraste con una conferencia, en la cual sólo una persona esta hablando, mientras otros escuchan. Existen diferentes maneras de controlar la interacción que se da en un grupo de trabajo:

- *Modo libre* , el cual es decidido mediante un acuerdo entre los participantes.
- *Basado en el sistema*, al contrario del anterior, es proporcionado por el mismo groupware.
- *Paso de estafeta*, el cual concede el turno al usuario que posee la estafeta.

La coordinación en un sistema groupware involucra el paso de mensajes que determinan la secuencia en que los eventos deben ser recibidos, atendidos y posteriormente difundidos.

Memoria organizacional. Se refiere al registro del proceso de la interacción del grupo, incluyendo la comunicación, las tareas ejecutadas, los productos obtenidos y su historial. Este tipo de

memoria proporciona a los colaboradores una mejor comunicación y coordinación en sus actividades.

Distribución. Gracias a las redes de computadoras, los usuarios, situados en diferentes lugares geográficos, pueden interactuar remotamente. De acuerdo a la forma en que los procesos están ubicados en una red, los sistemas groupware se pueden dividir dependiendo de su arquitectura en centralizados, distribuidos o híbridos.
En un sistema groupware centralizado existe un servidor central para controlar todas las entradas y salidas de los participantes. Los procesos clientes del sistema residen en cada sitio cliente. Cada uno de ellos, es responsable del envío y recepción de mensajes al servidor central. A su vez, cada sitio cliente se encarga del despliegue de las salidas enviadas por el servidor. La ventaja que ofrece esta arquitectura es la facilidad de manejar el estado consistente del sistema, ya que la sincronización de los procesos se lleva acabo en un sólo lugar; el servidor.

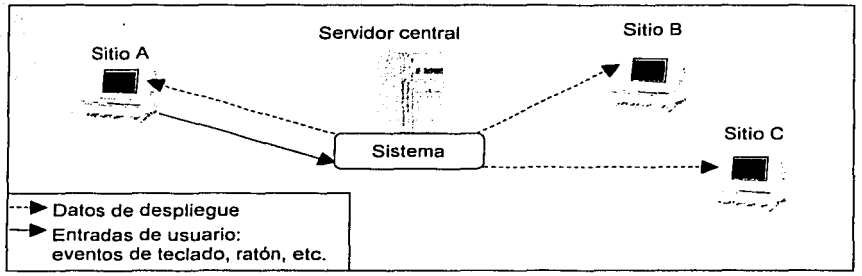


Figura 2.1. Arquitectura centralizada.

La figura 2.1 muestra un sistema síncrono con arquitectura de despliegue y manipulación de eventos en forma centralizada. Un cliente envía las entradas del usuario y los eventos que genera (con el teclado, ratón, etc.) al servidor central. Éste último los captura y mediante su sistema genera una salida, la cual es transmitida a los demás clientes.
En un sistema groupware distribuido, cada cliente posee y ejecuta una réplica (copia local) del sistema compartido. Las entradas y salidas de cada cliente son difundidas a todas las réplicas del sistema, las cuales se encargan de ejecutar los procesos necesarios para coordinar las acciones locales y remotas. Además, entre los clientes existe un proceso distribuido que se encarga de la sincronización de las réplicas para evitar estados inconsistentes en el sistema.
En la figura 2.2, se observa una arquitectura distribuida, en donde, cada sitio tiene un sistema replicado que se encarga tanto de la difusión de eventos locales como de la recepción de eventos remotos. Un factor importante a considerar en este tipo de arquitectura es el incremento de la complejidad, debido a la distribución de las réplicas del sistema, la sincronización entre éstas y el ordenamiento[27] de los eventos que ocurren en forma independiente a como deben ser procesados. Además, la difusión de eventos es otro factor que llega a ser un consumidor de tiempo, el cual debe ser consistente con el tiempo de despliegue de las interfaces gráficas de los clientes.
En la implementación de una arquitectura distribuida se sugiere que solamente se transmita, entre dichas copias, información respecto al estado del sistema. Así, mientras las actividades remotas pueden experimentar retardos, las actividades locales se ven ajenas a este problema y pueden ser

[27] Este mecanismo es importante para evitar problemas de *"lecturas pobres y escrituras prematuras"*.

atendidas inmediatamente. Con ello, se eliminan los cuellos de botella del procesamiento y mejoran el desempeño de la arquitectura centralizada.

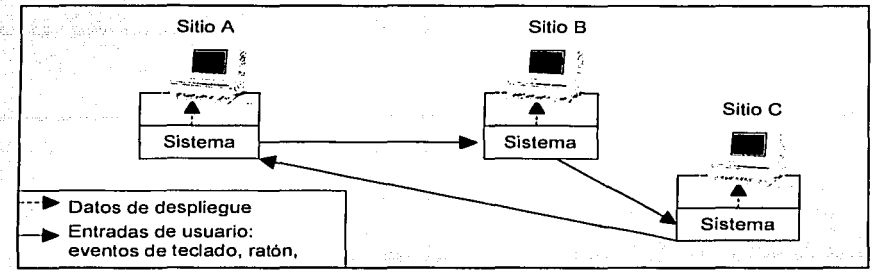


Figura 2.2. Arquitectura distribuida.

Por último, ambos tipos de arquitecturas han dado origen a modelos híbridos, en los cuales se incorporan procesos centralizados y distribuidos. Por ejemplo, el *X* Window de Unix es un sistema de difusión de despliegue, cuyo proceso servidor se encarga de tomar la entrada de los usuarios y hacerla llegar a las aplicaciones clientes, además de recibir y redireccionar las salidas de estas aplicaciones. Por lo general, el servidor *X* se ejecuta en la misma estación que las aplicaciones clientes, pero también se pueden ejecutar las aplicaciones clientes en máquinas remotas e interactuar con ellas en otra máquina que cuente con un servidor *X* (ver figura 2.2). También existe la posibilidad de que desde una única estación de trabajo con un servidor *X* se interactúe con aplicaciones gráficas que se ejecutan realmente en distintas máquinas.

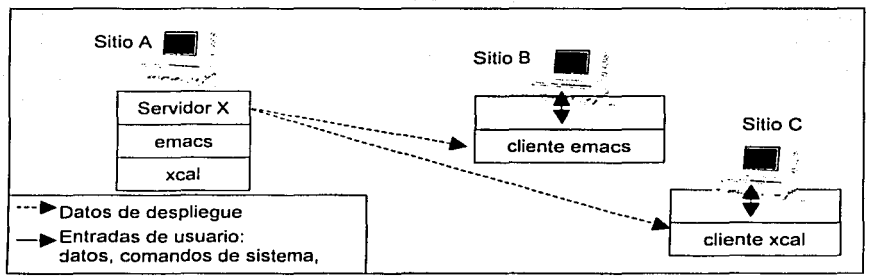


Figura 2.3. Arquitectura híbrida.

Flexibilidad. Se refiere a la habilidad de poder cambiar dinámicamente de estados o modos en un sistema groupware. Por ejemplo, en un sistema que tiene definido su espacio de trabajo pueden existir diferentes herramientas abiertas y el usuario puede estar interactuando con una y con otra sin que se vea alterada la ejecución de sus procesos.

Existe una alta flexibilidad en la visualización si se puede cambiar entre diferentes niveles de WYSIWIS. Por ejemplo, los usuarios pueden crear sus propias vistas para la compartición de datos, especificando si son públicos o privados.

En el aspecto de la interacción, se refiere a la posibilidad de cambiar entre modo síncrono y asíncrono en una sesión de trabajo colaborativo. Por ejemplo, en un sistema de edición

colaborativa, la planeación del documento se puede llevar a cabo en forma síncrona, mientras que la elaboración se realiza en forma asíncrona.
La flexibilidad en el aspecto técnico cubre la portabilidad y la adaptabilidad. Es preferible que el trabajo colaborativo se pueda llevar a cabo sobre diferentes arquitecturas de red, plataformas de hardware, sistemas operativos o sistemas de ventanas

Otros criterios. En particular, en esta tesis, se hará referencia a criterios técnicos como:

- *Manejo de fallas*, en el cual se trata de detectar problemas como fallas de energía eléctrica, comunicación en la red, datos corrompidos, entre otros, para ser resueltos o escondidos mediante el reemplazo de componentes. De esta forma se busca que el sistema siempre esté funcionando. Pero, si no es posible resolverlos, es conveniente enfocarse a la tolerancia de fallas, de tal manera que los usuarios sean advertidos del estado actual del sistema.
- *Desempeño*, es deseable que un sistema brinde tiempos de respuesta cortos entre los eventos que suceden y la percepción de los mismos.
- *Escalabilidad*, es otro criterio a considerar, ya que un sistema no debe verse afectado por el número de usuarios que participen en el trabajo colaborativo, aún si se encuentran trabajando en diferentes sitios geográficos.

En esta sección se han presentando algunos de los criterios más empleados en el diseño de sistemas de trabajo colaborativo. Sin embargo, pueden surgir otros dependiendo de las tareas que deba realizar el sistema groupware. Por ejemplo, la estandarización de los nombres de objetos que se crean en un espacio de trabajo, con el fin de identificar su tipo y la acción que pueden realizar.

2.2 Características de algunos sistemas groupware

Actualmente, existen diversos sistemas groupware que implementan espacios de trabajo compartidos. Estos espacios pueden ser directorios, escritorios, salones, pizarrones, oficinas y ambientes virtuales, permitiendo que los usuarios interactúen y utilicen simultáneamente los objetos existentes en el mismo.
Algunos de los sistemas que implementan espacios de trabajo compartidos son Mushroom [WMus], DeskTop [Por99], TeamWave [WTeW]; y en la Web, BSCW [WBSC] y GroupWeb [Gre97]. En la figura 2.4, se describen cada uno de ellos y también se mencionan las herramientas síncronas y asíncronas que incorporan. Los sistemas que basan su interacción mediante el manejo de escritorios y salones, son TeamWave, DeskTOP, Mushroom y Groove, los cuales, debido al diseño de sus espacios de trabajo y a la integración de una comunicación síncrona, proveen una mayor interacción y conciencia de grupo[28]. Por el contrario, BSCW basa su interacción en el manejo de directorios y la distribución de la información es totalmente asíncrona. Este sistema implementa una interfaz basada en los buscadores de páginas más comunes, por lo que resulta ser un sistema portable a la mayoría de las plataformas. De igual manera, GroupWeb se auxilia de la Web para realizar la distribución de la información, sin embargo su objetivo se centra únicamente en la visualización de documentos y anotaciones. Asimismo, la interfaz de DeskTop esta elaborada en lenguaje HTML, combinado con Applets de Java, lo que hace al sistema

[28] Se proporciona información sobre las ubicaciones actuales y acciones que los colaboradores acaban de ejecutar.

portable. Al contrario de las aplicaciones Web, los otros sistemas sólo se pueden ejecutar sobre una plataforma en particular, TeamWave en Unix y Windows, en el caso de GroupWise y Groove sólo en Windows.

Entre el groupware comercial se encuentra Groove y GroupWise. Sin embargo, este último es un sistema distribuido, cuyo principal servicio es el correo electrónico, que resulta muy costoso. La ventaja de las arquitecturas distribuidas es que ofrecen un sistema robusto y tolerante a fallas en comparación con las centralizadas.

Sistema	Arquitectura	Descripción	Herramientas	
			Síncronas	Asíncronas
Mushrooms	Centralizada	Soporta la administración y la creación de salones llamados Mrooms, en los que se comparten objetos.	Visualizador de actividades, chat, pizarrón blanco y manejo de video.	Referencias URL, visualizador de eventos, manejo de archivos, correo electrónico.
DeskTop	Híbrida	Se basa en los conceptos de salón, salas de colaboración y escritorios.	Chat, pizarrón, manejo de audio/video, teléfono, votación y editor.	Correo electrónico, notas, calculadora, calendario, votación y editor.
TeamWave Workplace	Centralizada	El servidor crea y mantiene cuartos permanentes y accesibles, donde la gente puede reunirse para trabajar y formar comunidades virtuales.	Manejo de archivos, pizarrón blanco, agenda, pizarrón de mensajes, votación y referencias URL.	Calendario, base de datos, creación de diapositivas.
GroupWise	Distribuida	Esta herramienta está definida por un dominio, junto con una oficina postal, una librería de documentos y uno o más usuarios.		Grupo de noticias, correo electrónico, programación de tareas, agenda, manejo de documentos.
Groove	Centralizada	Sistema basado en la tecnología COM que implementa espacios de trabajo compartido.	Chat, visualizador de páginas Web compartidas y pizarrón blanco.	Calendario, discusiones, Revisión de documentos, manejo de archivos, agenda y lluvia de ideas.
BSCW	Centralizada	Implementa espacios de trabajo compartidos basados en directorios y publicación de documentos en la WEB.		Manejo de documentos, direcciones URL, notas discusiones, calendario de reuniones y agenda.
GroupWeb	Híbrida	Es un visualizador de documentos WEB compartidos en tiempo real. Todos los visualizadores en sesión reciben una copia de la página original.	Visualizador de páginas compartidas , vistas WYSIWIS, barras de desplazamiento y telepuntadores.	Barras de desplazamiento, anotaciones a páginas.

Figura 2.4. Sistemas groupware.

2.3 Apoyo al desarrollo de sistemas groupware

En el campo de CSCW, se han propuesto herramientas de desarrollo para la implementación de sistemas groupware, conocidos como Toolkits. Estos se han enfocado en la solución de los problemas de la comunicación y la coordinación de actividades en la realización de una tarea en conjunto. De igual forma, estos sistemas abordan el problema de almacenamiento para

implementar la memoria de grupo y también, el problema de la distribución de objetos utilizados en el proceso de colaboración.

En la tabla de la figura 2.5 se describen algunos sistemas como son GroupKit [WGrK], COAST[29] [Sch96], COLA, NCSA Habanero [WNHa] y COCA[30] [LiMu98], cuyo principal interés es el desarrollo de sistemas síncronos. Se puede observar que cada Toolkit define su propia arquitectura, la cual también determina la plataforma de trabajo e incorpora diferentes mecanismos y herramientas para el desarrollo de groupware. Para el caso de GroupKit y Habanero, existen algunas herramientas de colaboración, por ejemplo el chat, el pizarrón blanco, la lluvia de ideas, la votación etc. Aunque estas herramientas no están incluidas en los demás Toolkits, se pueden desarrollar. Se muestra también que la mayoría de ellos permiten un desarrollo de programación orientado a objetos, es el caso de COAST, COLA , Habanero y COLA.

Sistema	Plataforma	Arquitectura	Servicios
GroupKit	W9x, Unix y Macintosh	Su arquitectura distribuida se basa en tres procesos: el registro, el administrador de sesión y la conferencia.	Soporta una comunicación síncrona en conferencias multi-punto e incoorpora un espacio de trabajo virtual y herramientas como lluvia de ideas, visualizador de archivos, creación de botones, chat y pizarrón blanco con tele-apuntadores.
COAST	WNT, W9x y Unix	Su arquitectura distribuida permite el desarrollo de aplicaciones cooperativas orientado a objetos.	Los principales servicios que ofrece son el control de transacciones para el acceso a objetos compartidos replicados , manejo de replicación transparente y control de concurrencia optimista.
COLA	Unix	Incorpora una arquitectura distribuida que permite la distribución de objetos compartidos.	Sigue un modelo de actividad ligera que permiten el desarrollo de tareas compartidas auxiliándose de mecanismos que incorporan componentes como actividades, papeles, objetos y eventos.
NCSA Habanero	W9x, WNT, W2K y Solaris	Incorpora una arquitectura centralizada, constituida por un servidor que organiza y conecta los procesos clientes en sesiones llamadas Hablets.	Permite el desarrollo de librerías para crear o convertir aplicaciones Java y applets en aplicaciones colaborativas. También proporcionan el ambiente para la creación de espacios virtuales de trabajo colaborativo e incluye herramientas como el telnet, el chat, la compartición de archivos, el pizarrón blanco y la votación.
COCA	Múltiples	Su arquitectura híbrida incorpora un modelo de comunicación de grupo basada en IP multicasting y una arquitectura de colaboración dual-bus para modelar las políticas de coordinación.	La actividad del colaborador es ejecuta en tiempo real por la máquina virtual de COCA. Esta también depende de los permisos que tenga el participante en el sistema.

Figura 2.5. Toolkits. Sistemas de desarrollo groupware.

Después de revisar algunos temas propios de los sistemas colaborativos, es conveniente señalar que este trabajo de tesis se enfoca en sistemas colaborativos que asistan la elaboración de documentos, por lo que en las siguientes secciones se aborda el tema de proceso de escritura, así como también, la descripción de algunos sistemas colaborativos que lo implementan.

[29] Del término en inglés *"Cooperative Application Systems Toolkit"*.
[30] Del término en inglés *"Collaborative Objects Coordination Architecture"*.

2.4 Edición colaborativa asistida por computadora

Dentro del campo del CSCW, la edición colaborativa se ha consolidado como una disciplina denominada *CSCWriting*, la cual ha aportado numerosas contribuciones enfocadas al desarrollo de sistemas y herramientas que facilitan la producción de documentos en forma colaborativa.
Ha existido un número de investigaciones sobre el proceso de la escritura. Entre ellas está la de encontrar cuál es la interacción entre planear, generar ideas y editar [HaFl80]. De igual manera, el de conocer cuáles son los eventos en el proceso de escritura, como las pausas [Mat81]. Así como, saber cuales son los procesos de composición de los escritores [Bri87]. Todos estos estudios han indicado que la escritura, la planeación y la generación de ideas están estrechamente relacionadas.
El proceso de escritura propuesto por [Sha94], se especifica un modelo llamado ciclo de creación y reflexión, en el cual se consideran cuatro etapas que son: la reflexión, la planeación, la creación y la revisión (ver figura 2.6). Este modelo se basa en el hecho de que mientras se escribe, se piensa lo que se escribe, pero se reflexiona sobre lo escrito cuando se hace una pausa y se lee. Este modelo postula que un autor tiene una serie de ideas que son organizadas para generar el texto. Al volverlo a leer, entrará otra vez al estado de reflexión, donde transformará sus ideas y llevará a cabo nuevas composiciones.

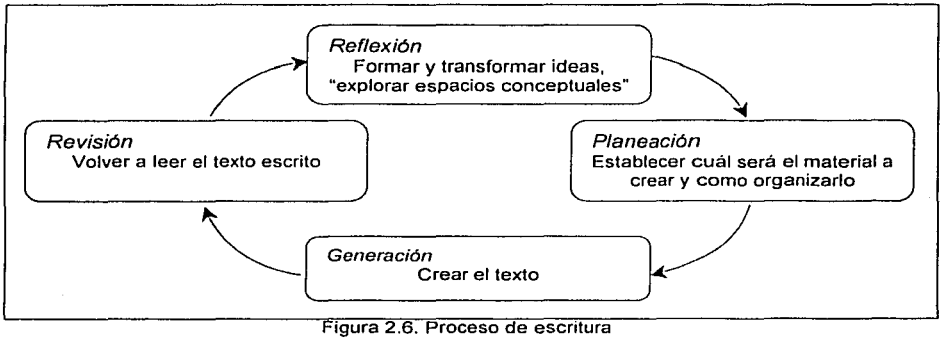


Figura 2.6. Proceso de escritura

Los investigadores [Bri87] y [Cha92] han descrito dos tipos de escritores: el planificador y el descubridor. El planificador es un escritor reflexivo que emplea mucho tiempo en la organización y el detalle del texto y, por consecuencia, no tiene un flujo de palabras muy fluido. En cambio, el descubridor cae en periodos muy largos de escritura, sin realizar una evaluación crítica. Sin embargo, ambos tipos de escritores pueden cambiar su enfoque de escritura, adaptándose a la forma más adecuada para dicha tarea.
Aunque la tarea de edición ha sido considerada como un esfuerzo individual en la que los escritores trabajan de forma aislada, la mayor parte de esta tarea implica una interacción social. Tanto en la industria como en la academia, la tarea de edición es realizada, en gran medida, por grupos de personas [Bai85]. Se han realizado una gran cantidad de estudios para investigar la manera en como las personas escriben documentos. Estos estudios están basados en observaciones, cuestionarios y entrevistas con grupos de escritores que han trabajado en diversos

proyectos, utilizando herramientas convencionales de edición. Los resultados de estos estudios muestran el interés de las personas por factores como: la planeación, la organización y la coordinación de las actividades colaborativas. Asimismo, sugieren la necesidad de otros factores, como la flexibilidad y la comunicación, durante el desarrollo de un proyecto de edición colaborativa. Existen varios modelos que describen la forma en que se lleva a cabo la edición. Sin embargo, el que más llama la atención, es el modelo de interacción social, debido a que califica al proceso de escritura, como un proceso concreto y social que afecta y se ve afectado por todas las personas involucradas. Este modelo, propuesto por [Nys89], describe al proceso de edición como una negociación entre el escritor y el lector, en la cual hay una interacción escritor-texto-lector. El escritor ya no es una persona aislada, ni el proceso de edición es abstracto e interno, por el contrario es un proceso concreto y social que afecta y se ve afectado por todas las personas involucradas. Cuando varias personas intervienen en el desarrollo de un documento compartido, la naturaleza social del proceso de edición se hace explícita. Por esta razón, la comunicación y la negociación de ideas se vuelven cruciales. El proceso de edición consta de tres etapas iterativas:

1. El escritor comienza un discurso escrito, es decir, forma el bosquejo de un texto con el fin de plasmar un conjunto de ideas relacionadas entre sí.
2. El escritor introduce nueva información y hace pruebas de correspondencia con el lector. Si la nueva información amenaza la correspondencia entre el escritor y el lector, entonces las ideas a transmitir no son del todo claras, por lo que representa una oportunidad para cambiar el discurso escrito. Sin embargo, puede que tanto el escritor como el lector defiendan sus diferentes puntos de vistas, los cuales deben desembocar en una buena solución para que no se originen malas interpretaciones de las ideas escritas.
3. En un intento por restaurar las ideas compartidas, el escritor emplea hipótesis tentativas de texto (también llamadas *elaboraciones*), para aclarar la comunicación con el lector.

El proceso de edición es considerado como un proceso mental y constructivo, es decir como una interacción entre las mentes de las personas implicadas, en vez de un proceso interno en la mente de un escritor aislado. Los problemas posibles que pueden surgir de este tipo de interacción son las incongruencias entre el escritor y el lector. Sin embargo, hay una forma de resolverlas, y esto es mediante las elaboraciones que el escritor propone como una clase de prueba para las respuestas del lector. El escritor y el lector trabajan conjuntamente para establecer un fundamento común estableciendo una comunicación.

2.4.1 Características de la edición colaborativa

Entre los estudios que han realizado [Kra90], [EdLu90], [PoBa92], [Bec93] y [BeBe93], se han conformado un número de modelos de edición colaborativa que describen la forma de trabajo de los grupos. Estos modelos se enfocan en la organización de los grupos, en roles, en las actividades y en las estrategias, con el fin de mostrar una variedad de factores inter-personales que tienen un impacto significativo en el proceso de edición. A partir de estos modelos, que a continuación se describen, se han diseñado e implementado los actuales sistemas de edición colaborativa.

2.4.1.1 Fases del proceso de edición colaborativa

[EdLu90] desarrollaron una clasificación de patrones de organización utilizados por los grupos (ver figura 2.7), con base en un estudio relacionado con personas involucradas en proyectos de edición colaborativa. Esta clasificación muestra los diferentes enfoques que toman los grupos con respecto a problemas de planeación, organización y división del trabajo, responsabilidad y revisión del documento.

No. patrón	Pre-escritura	Escritura	Re-escritura
1	El grupo planea y crea una tabla de contenidos.	Cada colaborador escribe una parte del documento.	El grupo combina las partes y revisa el documento.
2	El grupo planea y crea una tabla de contenidos.	Un colaborador escribe el documento.	El grupo revisa.
3	Un colaborador planea.	Escribe el documento.	El grupo revisa.
4	Un colaborador planea.	Escribe el documento.	Uno o más colaboradores revisan el documento sin consultar al escritor.
5	El grupo planea.	Escribe el documento.	Uno o más colaboradores revisan el documento sin consultar al escritor.
6	Un colaborador asigna actividades de edición.	Cada colaborador realiza una actividad individual.	Un colaborador combina las partes y revisa el documento.
7		Un colaborador dicta.	Otro colaborador transcribe el dictado y revisa el documento.

Figura 2.7. Patrones de organización utilizados por grupos [EdeLu90].

Observando las actividades de los participantes, en la elaboración de un documento, se ha determinado que el proceso de edición implica una serie de fases en las cuales los autores llevan a cabo sus actividades. En [PoBa92] [Neu94] y [Sha93], dichas fases se definen como:

1. **Fase de planeación**. Corresponde al inicio del proceso de edición. En esta fase, los autores establecen los objetivos, el contexto global de los documentos y su estructuración.
2. **Fase de escritura**. En esta fase, los autores realizan sus contribuciones generando diferentes estados del documento. Durante esta fase, cada autor se entera de las actividades que realizan los demás participantes del proceso.
3. **Fase de evaluación**. En esta fase, los autores realizan correcciones, comentarios y anotaciones sobre las contribuciones de sus demás colegas.
4. **Fase de negociación**. Corresponde a un período en el cual los autores discuten sobre las modificaciones que deben hacerse a los documentos. En ocasiones, esta fase se apoya de un proceso de votación para la toma de decisiones.
5. **Fase de consolidación**. En esta fase, los autores recopilan las estructuras de los documentos para su integración como un todo. Asimismo, los autores revisan los diversos cambios realizados y generan la versión final del documento producido.

No existe un orden secuencial de estas fases, pueden repetirse varias veces y en diferentes períodos hasta que se obtenga el documento final. El término autor se refiere aquellos participantes que tienen contacto directo (escribir, realizar notas o comentarios) con la edición del documento. Sin embargo, pueden existir otros colaboradores que realicen la actividad de sólo lectura.

2.4.1.2 Modos de interacción

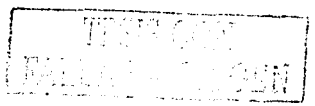
En todas las fases existe un contacto con todos los participantes. La comunicación directa se da cuando los participantes intercambian ideas o cuando trabajan sobre una parte específica del documento. Asimismo, existe una comunicación indirecta cuando cada autor realiza su contribución al documento en forma individual. Estos dos tipos de comunicación se refieren principalmente a los modos de interacción síncrona y asíncrona.
Como anteriormente se describió, en la sección 1.3.1, la interacción síncrona se refiere a que los participantes trabajan sobre un mismo recurso al mismo tiempo, en este caso, nos referimos al documento. Esta interacción comprende dos tipos [Bae94] [Rom98]:

1. **Interacción síncrona fuertemente acoplada.** Las contribuciones de los autores se integran al documento en tiempo real. Asimismo, el paradigma WYSIWIS esta presente, ya que todos los participantes perciben simultáneamente todas las acciones realizadas por cada autor, por lo que la vista del documento es la misma.
2. **Interacción síncrona débilmente acoplada**. Las contribuciones de los autores se integran al documento dependiendo de la voluntad del autor para difundir sus contribuciones y de la decisión de los participantes para aceptarlas. Debido a esto la visión del estado global de un documento puede no ser la misma para cada autor.

La interacción síncrona fuertemente acoplada tiene la ventaja de que los autores mantienen un único estado consistente del documento. Además, los autores pueden confrontar sus ideas en forma directa. Sin embargo, este tipo de interacción debe manejarse en forma apropiada, ya que los autores no deben verse afectados o interrumpidos por proceso de actualización constante como, por ejemplo, en la transmisión de un texto letra por letra.
La ventaja de la interacción débilmente acoplada es la de permitir trabajar, a cada autor, en forma aislada o simultáneamente en diversas secciones del documento. Sin embargo, se deben considerar mecanismos de sincronización y de control de versiones, para garantizar que cada uno de los participantes tenga una versión del documento consistente.
El modo de interacción asíncrona significa que cada autor trabaja sobre un documento en diferentes instantes de tiempo. Esta situación es muy común, ya que la mayor parte del trabajo de edición se lleva a cabo en forma asíncrona, donde las contribuciones se efectúan en forma individual e independiente de las acciones de los demás autores.
Una manera de enriquecer la funcionalidad de un sistema de edición colaborativa es haciendo coexistir ambas formas de interacción, permitiendo a los autores acoplarse según las actividades que desarrollen, otorgando una mayor flexibilidad en el proceso de edición.

2.4.1.3 Control del proceso de escritura

Existen diferentes esquemas para controlar la acción de escritura en un documento, los más comunes son: el control de documentos, la asignación de roles y el control de concurrencia.



Control de documentos

Los métodos de control de documentos describen la forma en que los colaboradores coordinan y realizan los cambios en un documento compartido. [BePo93] propone cuatro métodos para controlar los documentos:

1. **Centralizado.** Una persona controla el documento mientras que otras realizan sugerencias al escritor.
2. **Relevo**. Una persona controla el documento a la vez, pero el control pasa entre múltiples autores.
3. **Independiente**. Varias personas trabajan en segmentos del documento, pero cada uno mantiene el control sobre un segmento.
4. **Compartido**. Varias personas controlan el documento en conjunto, teniendo los mismos privilegios de escrituras.

Esta clasificación sugiere una división de responsabilidades entre los colaboradores. Hay que considerar que también las responsabilidades pueden cambiar durante el proceso de edición, según los requerimientos de la tarea.

Asignación de roles

La asignación de responsabilidades se refiere a la elección de los roles que juegan los colaboradores durante el proceso de edición, los cuales dependen de diversos factores como: la organización del grupo, las restricciones de tiempo de los colaboradores, las posiciones sociales, las habilidades y experiencias. [BePo93] identifica cinco roles entre colaboradores:

1. **Coordinador o administrador.** Organiza la división del trabajo y la asignación de roles.
2. **Escritor**. Convierte las ideas en texto y puede realizar las modificaciones sobre él.
3. **Consultor**. Participa en las diferentes actividades del proceso de edición, pero no escribe texto.
4. **Editor**. Realiza correcciones al texto escrito por alguien más.
5. **Revisor**. Realiza comentarios sobre el texto del documento.

Los roles han sido incorporados como mecanismos para controlar el acceso sobre los documentos. Por ejemplo, para garantizar que sólo un autor, cuyo rol es de escritor o de editor, puede agregar sus modificaciones en un instante de tiempo determinado. Los roles pueden cambiar en el proceso de edición para satisfacer los requerimientos de la tarea de edición.

Control de concurrencia

Existente diferentes mecanismos para el control de concurrencia y son implementados principalmente cuando se permite la elaboración de un documento de manera concurrente (múltiples autores trabajan sobre un documento al mismo tiempo), de tal manera que se eviten los conflictos que puedan surgir en la inserción, modificación y borrado de texto. Algunos de los mecanismos más utilizados para controlar la concurrencia son el bloqueo, el control de

concurrencia optimista y las marcas de tiempo, descritos en la sección 1.2.4 de este trabajo de tesis.

2.4.1.4 Arquitectura para el almacenamiento de documentos

Cada arquitectura de almacenamiento de documentos establece diferentes ubicaciones (sitios) para los clientes y los servidores, definiendo las operaciones para el acceso y la manipulación de los documentos. Las arquitecturas son centralizada, distribuida e híbrida.

Centralizada. Los documentos residen en un sitio servidor y los sitios clientes realizan peticiones al servidor para la edición del documento. Esta edición se realiza directamente sobre el documento que reside en el servidor, el cual utiliza un proceso para la coordinación y el control de las modificaciones que realizan los clientes.
Esta arquitectura ha sido utilizada ampliamente en los sistemas multiusuario tradicionales. Sin embargo, es una arquitectura que hace que el desempeño del servidor disminuya cuando crece el número de solicitudes clientes, es decir la carga de trabajo se incrementa. Además, si el servidor falla, el sistema queda en total inoperabilidad.

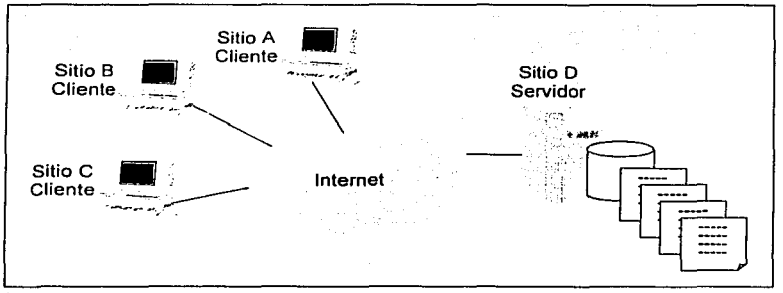


Figura 2.8. Arquitectura de almacenamiento centralizado.

En la figura 2.8 se muestran tres clientes, cada uno situado en diferente lugar geográfico. Los clientes trabajan sobre el documento almacenado en el sitio servidor. Los clientes no almacenan en forma local los documentos, sino que utilizan copias en memoria principal para trabajar con ellos.

Distribuida. En esta arquitectura, una copia (también llamada réplica) de cada documento es distribuida entre cada autor que participa en el proceso de edición. De esta forma, cada cliente trabaja sobre su copia del documento de manera aislada e independiente.
En cada cliente existe un conjunto de procesos para la distribución de las copias de los documentos y el tratamiento de su consistencia: cada autor puede modificar su copia local en diferentes instantes de tiempo y el sistema debe garantizar que el documento sea concebido como un todo, de forma coherente para todos los autores. Estas características sugieren que cada sitio cliente exista también un servidor de documentos y que cada autor puede crearlos y distribuirlos para que sean tratados en forma colaborativa.

Para el tratamiento de la consistencia de los documentos, copiados en cada sitio, se utilizan dos técnicas principales: el manejo de copia maestras y esclavas, y el manejo de versiones del documento.

La segunda técnica de consistencia se refiere a que en cada sitio se crean diferentes versiones de un mismo documento, a partir de sus copias distribuidas. La consistencia del documento se logra mediante la reunión de todas las versiones producidas y su comparación para determinar los cambios definidos. Para ello, se utiliza un mecanismo de negación entre los sitios, con el fin de resolver conflictos sobre los cambios realizados. Estos mecanismos varían dependiendo del nivel donde se aplique la comparación de las versiones, es decir, si la comparación se realiza sobre todo un documento, un párrafo, una línea o un caracter.

La arquitectura distribuida es muy apropiada para sistemas escalables y tolerantes a fallas. El desempeño del sistema no se ve afectado cuando el número de usuarios crece, puesto que cada uno trabaja sobre su copia local del documento. Asimismo, cuando existe una falla en algún sitio, ésta no afecta el resto del sistema debido a que cada uno trabaja en forma independiente.

La desventaja de esta arquitectura radica en que es necesario emplear un mecanismo para garantizar la consistencia del documento, el cual agrega mayor complejidad a las funciones de cada sitio. Además, dicho control de consistencia genera una mayor cantidad de mensajes entre cada sitio, lo cual agrega mayor tráfico a los canales de comunicación.

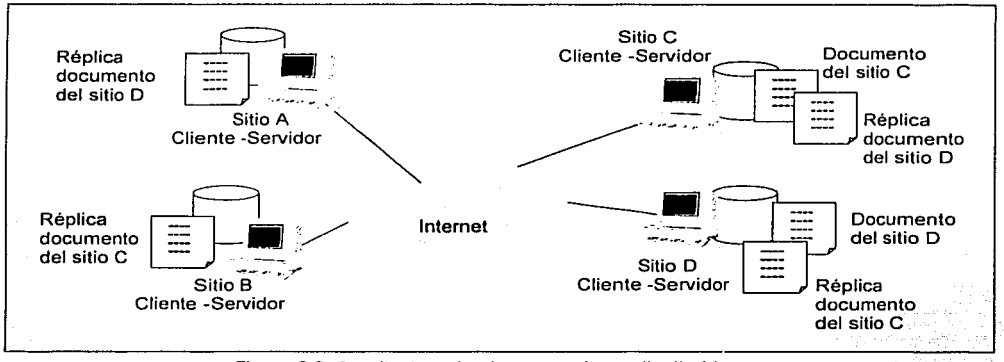


Figura 2.9. Arquitectura de almacenamiento distribuido.

En la figura 2.9 se muestra la distribución de documentos entre los sitios cliente-servidor A, B, C y D. El documento del sitio C, es copiado a los sitios B y D, mientras que el documento del sitio D es copiado solamente al sitio A y C. Además de copiar los documentos, es necesario que cada sitio difunda todos los cambios que se hagan sobre ellos.

Híbrida. Esta arquitectura combina las características de las arquitecturas centralizada y distribuida, por lo que puede darse en múltiples formas. Este tipo de arquitectura es la más usada en los editores colaborativos a gran escala. En cada ubicación existen procesos clientes y servidores al igual que en la arquitectura distribuida, pero las copias de los documentos se tratan de forma diferente.

Un ejemplo de este tipo de arquitectura se puede encontrar en la publicación de documentos en la Web. Los documentos se almacenan en forma centralizada en los sitios servidores y los sitios clientes obtienen una copia del documento mediante el método HTTP-GET. Las modificaciones del documento se realizan en los sitios clientes de manera local. Finalmente, utilizando extensiones del servidor de Web, mediante CGIs o modificaciones del método HTTP-PUT[31], los documentos son regresados de nuevo al servidor para su publicación. En la figura 2.10, se presenta la arquitectura básica de almacenamiento y publicación de documentos en la Web. Los sitios clientes A, B y D obtienen copias de los documentos almacenados en el sitio C, mediante el método HTTP-GET. Las modificaciones locales efectuadas al documento son enviadas al servidor mediante HTTP-PUT o ejecutando un CGI en el servidor. Estas extensiones son las responsables de mantener la consistencia del documento.

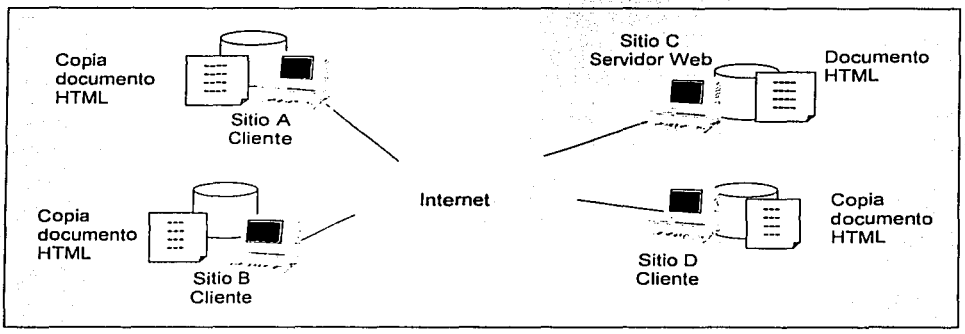


Figura 2.10. Arquitectura de almacenamiento distribuido.

Una extensión especial de la arquitectura híbrida es el modelo de tres capas ***cliente-broker-servidor***. En esta arquitectura se utiliza el almacenamiento centralizado, separando dos tipos de sitios servidores: para la atención de peticiones, que generalmente es el servidor de los procesos del sistema (servidor de aplicaciones) y para el control y el almacenamiento de los documentos en una base de datos (servidor de base de datos). Los sitios se diferencian como clientes y servidores, de los cuales algunos se distribuyen en servidores para la atención a los clientes y para la comunicación con el servidor de base de datos. De esta manera, cuando uno de los servidores del sistema falla, los clientes pueden ser direccionados a otros servidores para que el sistema continúe operando.

La figura 2.11 muestra una arquitectura típica de tres capas. Los sitios A y B realizan peticiones de documentos a los sitios servidores del sistema C y D. Nótese que en caso de una falla, en alguno de los servidores, el sistema puede seguir operando. Adicionalmente, el servidor de base de datos (sitio E) se encarga de las operaciones de control y bloqueo de los documentos almacenados, disminuyendo la carga en los servidores. Este modelo es muy utilizado en los sistemas distribuidos a gran escala, donde el factor de la disponibilidad y la tolerancia a fallas son muy importantes.

[31] Una modificación a este método la propone el grupo de investigación WEBDAV [Sus99].

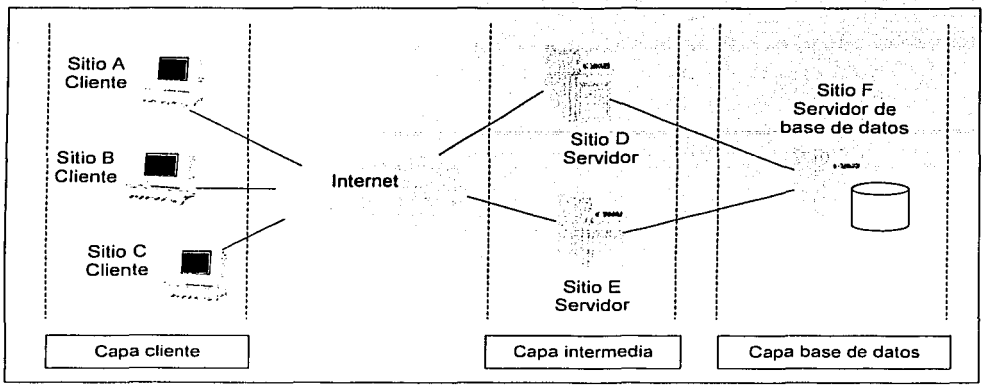


Figura 2.11. Arquitectura de tres capas.

2.5 Sistemas de edición colaborativa

En la actualidad, existen diferentes aplicaciones que permiten la elaboración de documentos de manera colaborativa. En la siguiente sección se muestran algunos sistemas de edición colaborativa en Internet más representativos que podemos encontrar en la literatura. Asimismo, para cada uno de ellos se describe con más detalle la manera en que controlan el proceso de escritura colaborativa, haciendo énfasis en el mecanismo de control de concurrencia que incorporan. Esta parte es importante resaltarla, ya que contribuye a la construcción de la propuesta de este trabajo de tesis.

2.5.1 GROVE ("Group Outline Viewing Editor")

Este sistema de autoría fue propuesto por [Ell91]. Su arquitectura híbrida está constituida por un editor local, replicación de documentos a cada estación de trabajo del usuario y un coordinador centralizado que serializa todas las operaciones de los diferentes editores.
Los documentos son desplegados en vistas grupales WYSIWIS mediante una estructura de árbol, donde los niveles de mayor jerarquía representan las divisiones del documento. Esta estructura es utilizada por los autores para asignar privilegios de lectura y escritura sobre las divisiones. Cada vista es distribuida en los sitios clientes y esta puede ser privada, compartida o pública. También se emplean múltiples vistas para la representación de las divisiones de cada documento. De igual forma, en la vista de grupo (ver figura 2.12), se representa el estado de interacción de cada autor con cada división.
El proceso de escritura se realiza en modo síncrono: cuando un usuario teclea un carácter, este texto llega a ser visible a los demás usuarios (notificación keystroke-level). Para controlar la concurrencia en este proceso, se recurre al mecanismo de *transformaciones de operación.* Éste consiste en que cada usuario tenga su propia copia del editor y cuando una operación es

solicitada, la copia se encarga de ejecutarla inmediatamente. Posteriormente, la operación es enviada en un mensaje de difusión junto con un vector de estado que indica cuantas operaciones ha procesado en otras estaciones. Cada editor tiene su propio vector de estado, el cual es comparado con el que llega. Si los vectores son iguales, la operación en el mensaje de difusión es ejecutada como petición; de otra manera es trasformada antes de la ejecución. La transformación depende del tipo de operación [Ell89].

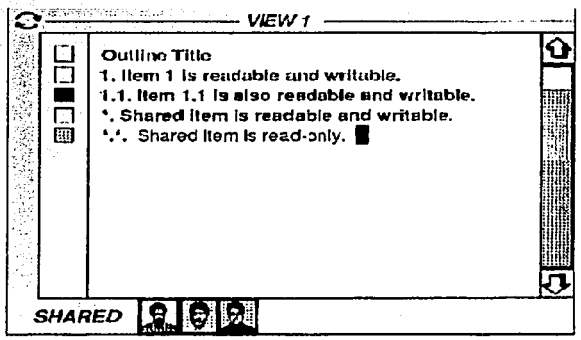


Figura 2.12 Editor GROVE.

2.5.2 PREP ("work in PREParation editor")

PREP [Neu90] es un editor colaborativo asíncrono que permite realizar anotaciones a un documento de manera síncrona. Su arquitectura híbrida consiste de tres unidades lógicas un agente fuente, un filtro y los receptores, de los cuales el agente fuente consulta al filtro para determinar que información es enviada a los receptores. El control de los documentos se lleva a cabo en una base de datos centralizada, donde además se establecen los filtros para la comunicación de eventos.

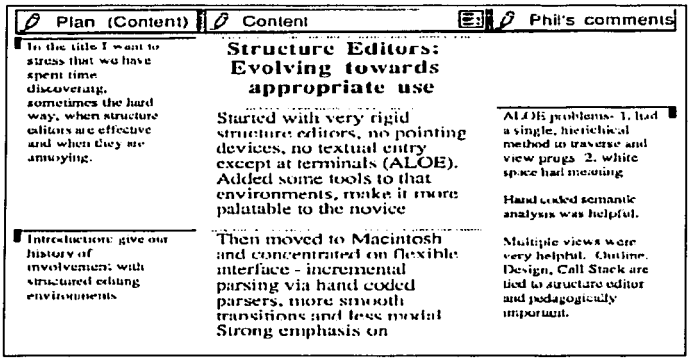


Figura 2.13 Editor PREP con tres columnas.

Cada documento es considerado como un conjunto de una o más columnas, las cuales, están constituidas por cadenas. Las primeras dos columnas se utilizan para la definición y planeación del documento (texto principal del documento) y para el contenido del mismo (ver figura 2.13). En las columnas siguientes se presenta en forma síncrona las anotaciones que cada autor realiza. El documento esta estructurado en forma de árbol, donde las hojas hacen referencia a cadenas que pueden contener la planeación del documento, contenido, anotaciones, imágenes u otras cadenas. Para controlar el proceso de escritura en un documento, las acciones de los autores se basan en *permisos*: limitado (lectura de algunas partes del documento), extendido (escritura en algunas partes del documento) y total (lectura, escritura y acceso total del documento). Sin embargo, cuando dos escritores escriben al mismo tiempo en el mismo párrafo, se puede originar un conflicto (pérdida de información), el cual puede ser solucionado mediante una interacción explícita.

2.5.3 MACE ("the Mother of All Concurrent Editors")

MACE [NePe91] es un editor, cuya arquitectura consiste de tres procesos: el administrador de archivos (FM) es el proceso de más alta jerarquía encargado del control de las sesiones. El administrador de editor (EM), es un proceso contenido en cada sesión, el cual maneja las peticiones de bloqueo y actualizaciones de archivo serializadas de manera FCFS (first-come-first-served), también maneja la serialización espacial de bloqueos lógicos que garantiza que no haya colisiones. Y una colección de ventanas editoras (EWs) que constituyen una arquitectura replicada parcialmente (ver figura 2.14).

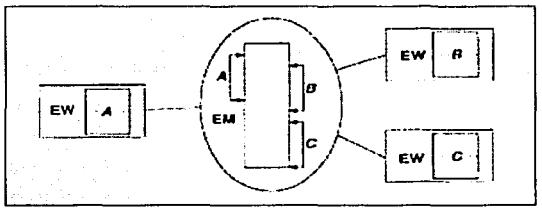


Figura 2.14. Editor MACE. Distribución de información en los sitios A, B, y C.

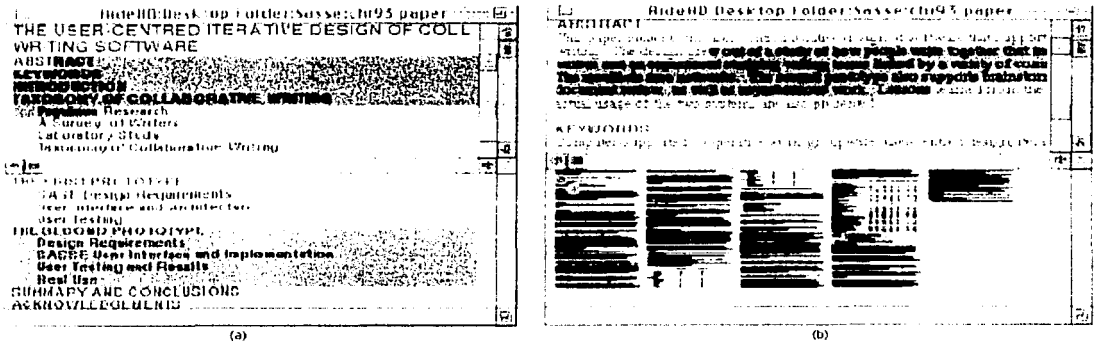
MACE proporciona tres modos de interacción para trabajar con los documentos. El primer modo, llamado de desplazamiento o estático, se presenta cuando un documento es desplegado en la ventana de edición. Cada usuario es libre de desplazarse, ver cualquier parte del archivo, y actualizar el documento cada vez que lo desee. Existe otro modo llamado actualización, en donde cada cambio hecho por un autor es reflejado a los demás espectadores en ese instante de tiempo. El último modo que maneja este editor es el síncrono, en el cual la frecuencia de actualización es en tiempo real y todos los cambios son inmediatamente reflejados.

Para escribir en el documento se debe adquirir un par de bloqueos (inicio y final) que involucran la sección del texto a ser editado. Estos bloqueos particionan lógicamente el documento para conceder derechos exclusivos de escritura. El usuario puede salvar o deshacer sus cambios en cualquier tiempo durante la edición. Una vez realizados los cambios, la actualización remota

aparecerá en la pantalla de todos los otros usuarios con la sección cambiada del texto entre sus vistas si se encuentran en modo de actualización.
Este editor recurre al mecanismo de bloqueo para controlar la concurrencia en el proceso de escritura. Cuando el bloqueo es solicitado, el EW revisa si existen localmente colisiones. Si este no es el caso, se envia una solicitud al EM que se encarga de verificar que no haya colisiones en la tabla de bloqueo central, posteriormente, determina si otorgar o no el bloqueo. Si el bloqueo es otorgado, se informan a todos los editores afectados mediante un mensaje de difusión.

2.5.4 SASSE ("Synchronous Asynchronous Structured Shared Editor")

Este sistema de edición colaborativa propuesto por [Bae93] se basa en la organización estructurada de las actividades de edición. En su arquitectura replicada existe un servidor de comunicación centralizado el cual asegura que todas las copias de las aplicación reciban mensajes en el mismo orden. Además, asegura que la copia de la aplicación y el documento compartido residan en cada estación de trabajo del colaborador.
Los autores colaboran sobre una vista WYSIWIS, sincronizada del documento, la cual es común para todos los colaboradores. Se usan secciones coloreadas y un cursor remoto (telepuntero) en modo síncrono, para indicar la ubicación de cada autor en el documento compartido. Asimismo, incorpora una barra de desplazamiento, la cual indica mediante un color en particular, el lugar en que cada autor está trabajando actualmente en el documento (ver figura 2.15). Soporta también lluvia de ideas, crea perfiles, vistas de la estructura de un documento y procesos de revisión.



(a) (b)

Figura 2.15. Editor SASSE. (a) Vista de la sección editada del documento y la actividad de otro usuario en particular. (b) Vista del documento editado y las ubicaciones de los demás autores en el documento.

El editor SASSE provee los mismos derechos de lectura y escritura a todos los autores, por lo que cada uno de ellos puede modificar y agregar texto en cualquier parte del documento. Sin embargo, como el proceso de escritura se realiza en modo síncrono, puede existir la posibilidad que dos o más autores estén trabajando sobre una misma sección del documento. Para resolver los problemas de sobreescritura y pérdida de la información que se pueden dar en esta situación, SASSE utiliza el mecanismo de *bloqueo* sobre texto seleccionado, limitando a que un sólo autor pueda trabajar en él.

2.5.5 DUPLEX

La arquitectura de este editor colaborativo propuesto por [PuSa94] es distribuida y solamente puede ser ejecutado en plataformas UNIX usando el sistema de ventanas X11 y el protocolo TCP/IP. El modelo DUPLEX incorpora tres conceptos básicos. El primero se refiere a la descomposición del documento, el cual especifica un conjunto de reglas para dividir un documento en partes independientes, esta estructura es establecida por los autores. El segundo concepto es la existencia de un núcleo, compartido por todos los miembros de la colaboración, el cual proporciona la persistencia y disponibilidad de los más recientes documentos. El núcleo está constituido por objetos, llamados kernel que definen operaciones de lectura, actualización, división etc., y son replicados para asegurar la disponibilidad y el rápido acceso.
Por último, el ambiente del usuario local, el cual proporciona un almacenamiento personal y un espacio de trabajo autónomo para cada autor. Además contiene un conjunto de copias de objetos kernel que componen la vista local del documento. Cada vez que se realiza una operación sobre los objetos kernel, desde el ambiente local del usuario, se realiza una conexión temporal al núcleo. Y cuando es completada la operación, el ambiente del usuario se desconecta del núcleo. Una consecuencia del modelo DUPLEX es que las vistas de los documentos de los usuarios pueden ser diferentes de aquellas replicas mantenidas por el núcleo. En la figura 2.16 se visualiza la estructura de un documento en segmentos.

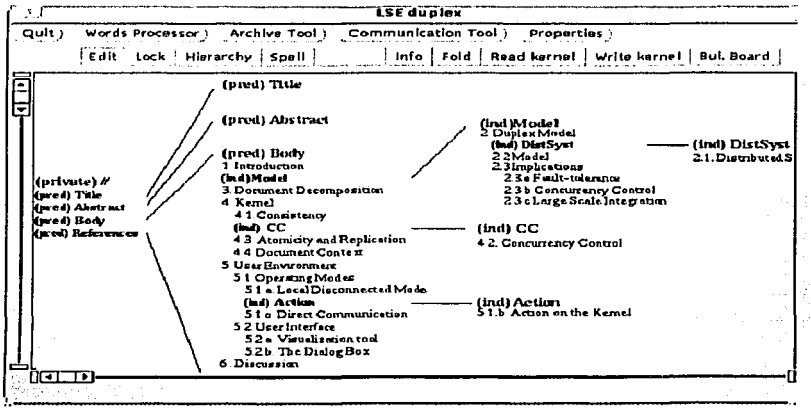


Figura 2.16. Visualización de la estructura del documento en segmentos

DUPLEX considera tres enfoques en sus políticas para controlar la concurrencia en el acceso de datos compartidos: la unidad de concurrencia del documento (segmentos), responsabilidades de los usuarios sobre los segmentos y la confianza de la percepción del usuario ante un conflicto. A partir de estos enfoques los autores podrán definir que tipo de control de concurrencia desean integrar en la parte en la que están trabajando: pesimista (de acuerdo a los roles) u optimista (registros replicados lineales).

2.5.6 ALLIANCE

El editor colaborativo ALLIANCE [Rom97] permite que diversos autores, ubicados en sitios geográficos diferentes, puedan trabajar conjuntamente sobre documentos estructurados. ALLIANCE utiliza una arquitectura distribuida para la administración y el almacenamiento de los documentos de trabajo, mediante el manejo de copias de los documentos y los fragmentos, en cada sitio que participa en el proceso de edición. Adopta parcialmente la arquitectura de la Web para su funcionamiento, explotando ampliamente el modelo cliente-servidor y los métodos GET y POST del protocolo HTTP. Sin embargo, implementa su propia interfaz, basada en el editor THOT y maneja los documentos codificados en un formato específico denominado pivote.
La edición colaborativa se basa en tres principios fundamentales: manejo de roles de edición (lector, escritor, administrador y nulo), estructuración de documentos (división del documento en unidades básicas conocidas como fragmentos) y apoyo a la conciencia de grupo, en donde cada autor trabaja sobre su propio ambiente particular, percibiendo la evolución de las contribuciones de los demás coautores y controlando la difusión de su propia contribución. Aunando a esto, los documentos se presentan en vistas WYSIWYG con iconos representativos de roles y acciones permisibles. En la figura 2.17 se muestra la asignación de roles para los colaboradores sobre el fragmento que esta seleccionado

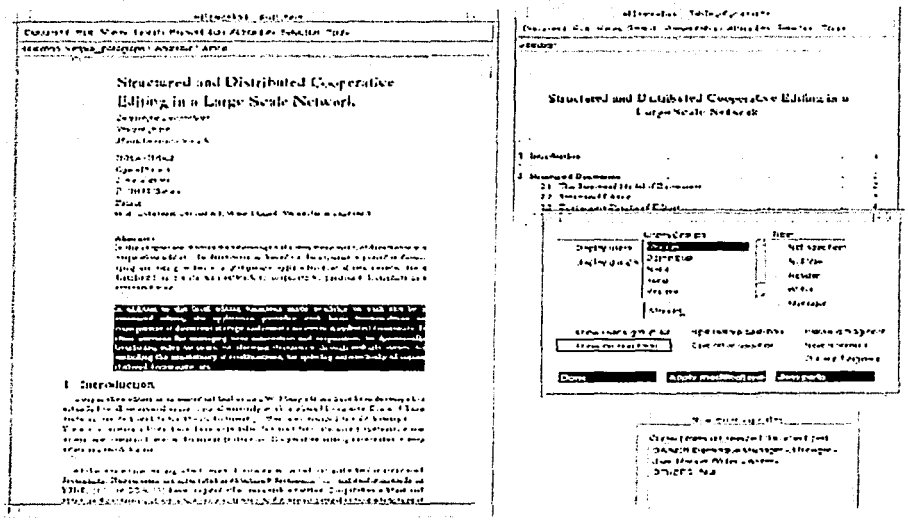


Figura 2.17. Asignación de roles en ALLIANCE.

Para el control del proceso de escritura en un fragmento, utiliza el mecanismo de bloqueo. De tal manera que sólo exista un escritor potencial (aquel que adquirió el bloqueo) y muchos lectores al mismo tiempo en un mismo fragmento.

2.5.7 MPEDIT

El editor MPEDIT es una aplicación que se encuentra incluida en el sistema groupware llamado HABANERO. En él, existe un servidor centralizado que se encarga de la administración de los documentos y de la serialización de las operaciones de edición (cortar, copiar y pegar). La colaboración que se puede realizar en el editor es mínima debido a que un colaborador debe obtener el bloqueo de todo el documento para realizar la acción de escritura. De esta manera un participante edita el texto a la vez. El documento puede residir en la parte cliente o en el servidor. El tipo de colaboración que se realiza en este editor no trabaja bien con la dinámica de un grupo activo, además no existe el paso de información para fomentar la conciencia a otros usuarios.

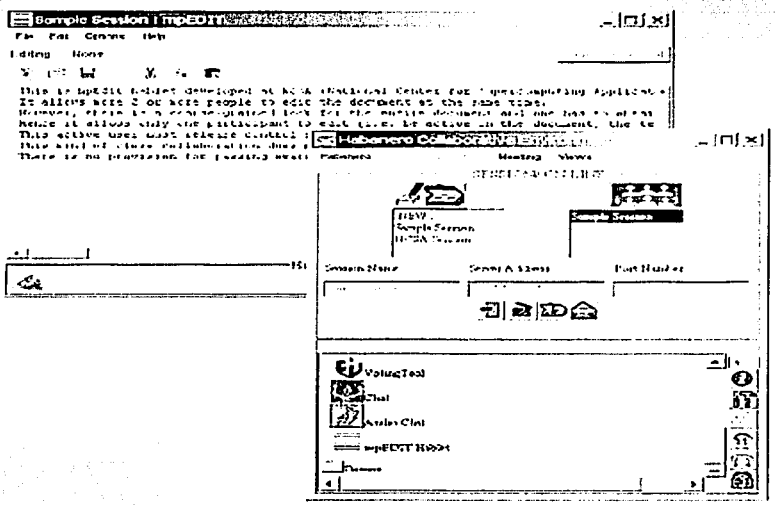


Figura 2.18. Editor MEDIT.

En la figura 2.18, se visualiza en primer plano la ventana de HABANERO, la cual contiene una serie de herramientas groupware. A partir de esta ventana se llama a la aplicación MPEDIT, esta se muestra en un segundo plano.

2.5.8 COARSY ("Collaborative Asynchronous Review System")

COARSY [RuFa98] es un sistema colaborativo asíncrono para la edición de documentos HTML. Este sistema esta desarrollado en JAVA y su arquitectura híbrida consta de tres capas. El cliente es una aplicación independiente que los usuarios usan para revisar y modificar el documento. En la parte servidora, existe un servidor de base de datos para administrar toda la información de los documento y los usuarios. Así también, hay una aplicación servidor usada para comunicarse con el servidor de la base de datos y obtener o proporcionar la información requerida por los clientes activos para la ejecución de sus operaciones.

COARSY se basa en el modelo conversacional, en el cual los componentes de una discusión (preguntas, respuestas, anotaciones, comentarios) y sus relaciones, permiten la estructuración y la revisión de un documento en forma colaborativa. Ver figura 2.19.
Los autores realizan sus contribuciones únicamente en la sección de discusión, debido a que el sistema no permite escrituras concurrentes. Solamente el autor dueño del documento es quien realiza las respectivas modificaciones, apoyado en las discusiones generadas con los demás coautores. De esta forma, se establecen los roles de autor (escritor) y revisor (contribuye a la discusión), para facilitar su interacción y determinar los accesos a las discusiones que se llevan a cabo durante la producción del documento. Debido a que la elaboración del documento lo realiza sólo un autor, COARSY no incorpora algún mecanismo de control de concurrencia.

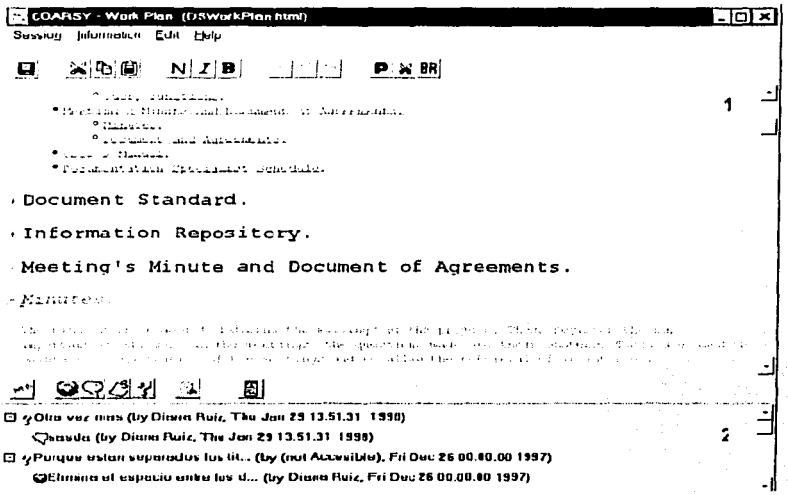


Figura 2.19. Editor COARSY. (1) Visualización del documento. (2) Área de discusión.

2.5.9 D3E ("Digital Document Discourse Enviroment")

Este sistema propuesto por [TaSi98], esta diseñado para la publicación de documentos Web, el cual ha incorporado algunas funciones de discurso y ciertos componentes que permiten la interación. Este consisten de un conjunto de herramientas destinadas a la creación y administración de un sitio, en donde residen las plantillas (archivos) que contienen variables específicas D3E y que indican donde se deben almacenar los diferentes tipos de documentos. Para la generación de documentos se hace uso de una aplicación Java llamada Toolkit Publisher que toma como entrada archivos HTML y realizar sobre cada uno de ellos un análisis sintáctico para crear archivos HTML con formas especiales de navegación y funciones de discursos. Todo este proceso es realizado localmente en la computadora del escritor. Cuando éste considera que el documento ya esta listo para ser leído por otros colaboradores, lo debe colocar en un servidor Web y anunciar su locación.

Para crear la interfaz del documento, se hace uso de una versión modificada de HyperNews [WHyn], la cual, incorpora las funciones de discurso. En la figura 2.20, se muestra la salida del artículo hecho con el Toolkit de D3E. En la parte de la izquierda se muestra la ventana del artículo y en la derecha la ventana de comentarios mostrando las líneas de discusión acerca del documento. En la sección 1 se muestran el icono que permite desplegar los comentarios. En la 2, la lista de contenido activa. Asimismo, en la sección 3 se muestra un icono para bajar la versión de Acrobat. La sección 4, muestra una cita ligada automáticamente a la ventana de pie de página (sección 5). En la sección 6 se despliegan las ligas en retroceso en el artículo. La sección 7 muestra la discusión general de encabezados definidas en el toolkit. Finalmente, la sección 8 se encuentran los encabezados para comentarios de una sección específica.

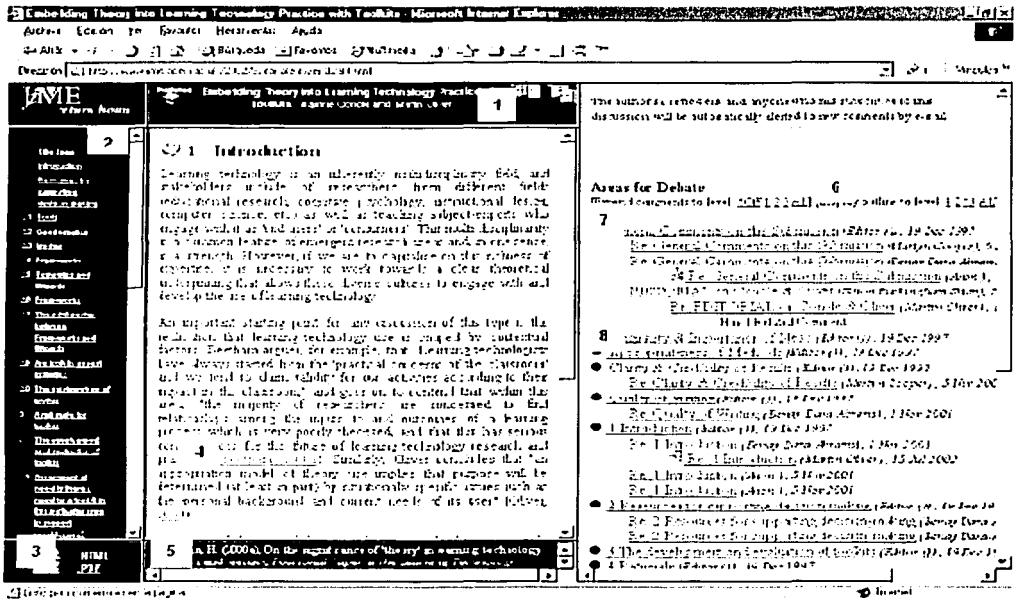


Figura 2.20. Salida de un documento de D3E en HTML.

2.6 Comparación de sistemas de edición colaborativa

Con el fin de mostrar cuales han sido las aportaciones más relevantes de cada uno de los editores de la sección anterior, se hará una comparación entre ellos. Para ello, se han considerando los criterios de conciencia de grupo, fases de proceso de edición, los modos de interacción, la arquitectura de almacenamiento de documentos y el control del proceso de escritura.

Como se puede observar en la tabla de la figura 2.21.a, los sistema de edición GROVE, MACE y SASSE apoyan la conciencia en grupo incorporando en el espacio de trabajo vistas WYSYWIS, las cuales muestran a los usuarios los cambios hechos en los documentos en tiempo real. Sin

embargo, de estos editores, únicamente MACE incorpora otro modo de vista que es el WYSIWIMS, que aunque no se tiene una versión actualizada del documento, el autor no es interrumpido por las actualizaciones realizadas por otros colaboradores. Por otra parte, en PREP, DUPLEX, ALLIANCE y MPEDIT sólo proporcionan vistas WYSIWYG, en las cuales no se visualizan de manera síncrona, las modificaciones que se lleven a cabo en un documento.
Casi todos los editores listados en la tabla de la figura 2.21.a, integran principalmente las fases de planeación y escritura de documentos. Sin embargo, es posible notar que COARSY y D3S tiene la fase de escritura individual porque no permiten la edición concurrente, pero la contribución de los lectores se realiza mediante discusiones apoyadas por el registros de comentarios. Por otra parte, GROVE, SASSE, MACE y ALLIANCE proponen que el proceso de escritura sea en modo síncrono, a diferencia de PREP, DUPLEX y MPEDIT. De acuerdo a un estudio realizado [PoBa92] acerca de la escritura colaborativa, se concluyó que las estrategias de edición síncrona y asíncrona son usadas en diferentes fases de un proyecto de escritura colaborativa. Por lo tanto, es recomendable que un sistema de edición colaborativa soporte ambos modos de comunicación: síncrono y asíncrono.

Herra-mienta	Conciencia de grupo	Fases de proceso de edición	Interacción
GROVE	Vistas WYSIWIS grupales, públicas y privadas.	Planeación y escritura colaborativa.	Establece una escritura síncrona débilmente acoplada. Y la visualización es síncrona.
PREP	Vistas WYSIWYG y columnas de anotaciones.	Planeación, escritura y consolidación colaborativa.	El modo de escritura es asíncrono y anotaciones en forma síncrona débilmente acoplado,
MACE	Vista WYSIWIMS en el modo estático y vista WYSIWIS en el modo de actualización y síncrono. Ventana de discusión en modo síncrono.	Planeación, escritura y consolidación colaborativa.	La edición puede realizarse en modo síncrono y asíncrono.
SASSE	Vistas WYSIWIS, telepuntero y barras de estado y áreas de texto marcadas con colores diferentes para cada usuario.	Planeación, escritura y consolidación colaborativa.	El modo de edición puede ser asíncrono y síncrono débilmente acoplado. La comunicación entre usuarios es en modo síncrono.
DUPLEX	Vistas WYSIWYG.	Planeación y consolidación individual. Escritura colaborativa.	La escritura se realiza en modo asíncrono.
ALLIANCE	Vistas WYSIWYG e iconos que representan roles y acciones permisibles.	Planeación y consolidación individual. Escritura colaborativa.	La escritura se realiza en modo asíncrono y síncrono debilmente acoplado.
MPEDIT	Vistas WYSIWYG	Escritura individual.	La escritura es asíncrona
COARSY	Iconos de discusión en forma síncrona y registro de las contribuciones a las discusiones.	Planeación colaborativa y escritura individual.	Solamente se llevan a cabo discusiones y anotaciones asíncronas.
D3S	Registro de comentarios	Planeación colaborativa y escritura individual.	Solamente se llevan a cabo discusiones y anotaciones asíncronas.

Figura 2.21.a. Características generales de sistemas de edición colaborativa.

Uno de los factores importantes en el diseño de la arquitectura de un sistema de edición colaborativa es establecer la forma de almacenamiento y acceso a los documentos. Por ejemplo,

SASSE, DUPLEX y ALLIANCE establecen una arquitectura distribuida, cuya ventaja es eliminar los cuellos de botella por procesamiento, siendo notables en sistemas centralizados, o bien el grado de tolerancia de fallas que manejan cuando existen situaciones donde se desea cubrir la vulnerabilidad de los medios de comunicación. Otra ventaja que presentan DUPLEX y ALLIANCE es la edición de documentos en modo sin conexión, la cual permite que los colaboradores realicen una acción en el sistema sin la necesidad de enviar mensajes al servidor.

Herramienta	Arquitectura	Estrategias de edición		
		Control de documentos	Roles	Control de concurrencia
GROVE	Híbrida	Independiente	No son explícitos, sin embargo, los autores pueden acceder al índice de un documento vía derechos de acceso.	Transformación de operación.
PREP	Híbrida	Independiente	Los roles son co-autor, lector y escritor.	
MACE	Híbrida	Compartido	No son explícitos, sin embargo, todos los colaboradores tiene derechos de lectura y escritura .	Bloqueo de escritura sobre fragmento
SASSE	Distribuida	Compartido	No son explícitos, sin embargo, todos los colaboradores tiene derechos de lectura y escritura .	Bloqueo sobre texto seleccionado
DUPLEX	Distribuida	Compartido	No son explícitos, sin embargo todos los colaboradores tiene derechos de lectura y escritura.	Bloqueo y control de concurrencia optimista.
ALLIANCE	Distribuida	Independiente	Existen cuatro tipos de roles sobre cada fragmento: lector, escritor, administrador y nulo.	Bloqueo sobre fragmento
MPEDIT	Centralizada	Relevo	No son explícitos	Bloqueo sobre el documento
COARSY	Híbrida	Centralizado	No provee un mecanismo para hacerlos explícitos, sin embargo existe un autor y varios revisores.	
D3S	Centralizada	Centralizado	No provee un mecanismo para hacerlos explícitos, sin embargo, existen un autor y varios revisores.	

Figura 2.21.b. Características generales de los sistemas de edición colaborativa.

Las estrategias de edición han surgido con el fin de coordinar las acciones de los colaboradores en un documento compartido. Éstas se pueden llevar a cabo implementando algún esquema de control de documentos, así como la asignación de responsabilidades, mediante roles a los colaboradores y la incorporación de un mecanismo de control de concurrencia. No todos los sistemas listados en la tabla de la figura 2.21 incluyen estas estrategias. Por ejemplo, el control de documentos en COARSY y D3S es centralizada, es decir, sólo existe un escritor que acepta los comentarios de otros colaboradores, por lo que no existen ningún problema en el proceso de edición. Por lo tanto, no requiere ningún mecanismo de control de concurrencia. El editor MPEDIT incorpora el control de documentos por relevo, en donde un solo autor puede escribir en él cuando lo solicita y cuando lo libera puede ser adquirido por otro autor. En el caso de los editores MACE, SASSE y DUPLEX, el control de documentos se realiza de manera compartida y aunque no hacen

explícitos los roles para los colaboradores, la concurrencia en el proceso de escritura la controla mediante un mecanismo propio. GROVE, PREP y ALLIANCE controlan los documentos de forma independiente, cada documento es seccionado en unidades y en cada una de ellas los colaboradores tienen asociados roles o permisos.

Cada editor establece las actividades que un autor puede realizar en el documento mediante la asignación de roles. En PREP además de los roles de lector y escritor, incorpora otro llamado co-autor, cuya actividad es la de revisar y hacer comentarios al documento. Este último rol no lo define ALLIANCE, sin embargo define el de administrador y nulo, debido a la forma en que estructura el documento. Por otra parte, GROVE aunque no hace explícitos los roles de los colaboradores, si maneja permisos de acceso.

Los mecanismos de control de concurrencia son incorporados principalmente por aquellos sistemas que permiten la edición concurrente. DUPLEX, MACE y ALLIANCE utilizan el mecanismo del bloqueo de escritura, de tal forma que el fragmento o sección del documento sea exclusivo para un sólo autor en el proceso de escritura. De igual manera, SASSE utiliza el mecanismo de bloque; pero sobre texto seleccionado, haciendo que la edición sea de grano más fino que los anteriores. Aunque MPEDIT incorpora el bloqueo como mecanismo de control de concurrencia, no es tan eficiente en grupos de colaboración dinámica, ya que lo realiza sobre todo el documento. Otro mecanismo que proporciona DUPLEX es el control de concurrencia optimista, mediante el manejo de registros replicados lineales que se encargan principalmente del ordenamiento de las operaciones. A pesar de ser menos caro que el bloqueo, todavía no resuelve problemas de sobre-escritura ni actualización. Con respecto a GROVE, este editor utiliza un mecanismo llamado transformación de operación orientado a resolver problemas de edición síncrona.

2.7 Resumen

El Trabajo Colaborativo Asistido por Computadora (CSCW) se ha convertido en un campo de estudio de sumo interés para aquellos interesados en el trabajo en grupo con ayuda de las computadoras, en el diseño del software y el comportamiento social y organizacional. Como parte de una necesidad dentro de esta disciplina, han surgido los sistemas groupware, los cuales han sido diseñados contemplando ciertos criterios y requerimientos. Algunos ejemplos de sistemas groupware que se vieron en este capítulo son: Mushrooms, DeskTop, TeamWave, BSCW y GroupWeb, los cuales implementan su propio espacio de trabajo, así como una serie de herramientas que fomentan la conciencia en grupo (niveles de vistas WYSIWIS, telepunteros, registro de actividades, comunicación síncrona, compartición de información etc.). Es importante destacar que cada sistema groupware implementa una tarea específica. Por ejemplo, GroupWeb se dedica a la publicación de documentos, GroupWise al sistema de correo electrónico y BSCW en la compartición de la información mediante archivos. Sin embargo, algunos de ellos implementan otras herramientas (agendas, registro de comentarios, etc.) para poder llevar a cabo satisfactoriamente su tarea.

Para apoyar el desarrollo de este tipo de sistemas han surgido los Toolkits (herramientas de desarrollo), enfocados principalmente a la solución de problemas de comunicación y la coordinación de actividades en la realización de una tarea en conjunto. Ejemplos de ellos son: GroupKit, COAST, COLA, NCSA Habanero y COCA, los cuales definen su propia arquitectura, mecanismos y herramientas para el desarrollo de sistemas groupware. Algunas desventajas de

este tipo de Toolkits es que tiene predeterminada la plataforma de trabajo, que en ocasiones no lo hace tan portable, este es el caso de COLA. Asimismo, para el desarrollo de la interfaz gráfica se requiere una mayor programación en GroupKit y COLA.

Dentro del campo del CSCW, la edición colaborativa se ha consolidado como una disciplina denominada *Edición Colaborativa Asistida por Computadora (CSCWriting)*, la cual ha aportado numerosas contribuciones enfocadas al desarrollo de sistemas y herramientas que facilitan la producción de documentos en forma colaborativa. Se han realizado una gran cantidad de estudios para investigar la manera en que las personas escriben colaborativamente. Estos estudios están basados en observaciones, cuestionarios y entrevistas con grupos de escritores que han trabajado en diversos proyectos, utilizando herramientas convencionales de edición. Los resultados de estos estudios muestran el interés de las personas por factores como la planeación, la organización y la coordinación de las actividades colaborativas. También se han propuesto un número de modelos de edición colaborativa que describen la forma en que los grupos se organizan para trabajar. Por lo tanto, estos modelos se enfocan en la organización de los grupos, en roles, en las actividades y en las estrategias.

Algunos de los editores colaborativos más representativos en la literatura son: GROVE, PREP, MACE, SASSE, ALLIANCE, MPEDIT, COARSY y D3S, cada uno de ellos posee sus propias características de interacción social y procesos de edición. Uno de los criterios que se emplea para el diseño de sistemas de autoría colaborativa es el de conciencia de grupo, que regularmente es implementada por: diferentes niveles de vistas WYSIWIS del documento, telepuntero, barras de estado, iconos que representan roles y acciones, o bien registro de comentarios. Otro criterio es el de la interacción, cuyo modo puede variar. Hay sistemas que únicamente implementan la edición síncrona (GROVE), la asíncrona (PREP y MPEDIT), o bien, ambos tipos (SASSE, MACE, DUPLEX y ALLIANCE). Pero existen otros que están enfocados totalmente a la contribución de comentarios (COARSY y D3S).

En la arquitectura de un sistema de edición pueden existir diferentes mecanismos implementados para el control de documentos. Sobre todo, en aquellos sistemas que apoyan la edición concurrente. Algunos editores proponen esquemas basados en responsabilidades, mediante roles o permisos, o bien, mecanismos de control de concurrencia (por ejemplo bloqueo de escritura sobre fragmentos, bloqueo sobre texto seleccionado, transformación de operaciones, etc.). Sin embargo, existe la posibilidad de que no sea suficiente implementar uno de estos mecanismos. Por ejemplo, PREP sólo implementa el esquema de roles, el cual no resuelve los conflictos que surgen cuando varios colaboradores tratan de editar la misma parte del documento.

El control de la concurrencia, en un editor de sistema colaborativo, es el tema principal de este trabajo de tesis y será abordado en el siguiente capítulo en un nuevo sistema de edición llamado *Editor coLaborativo de documentos XML en Internet (ELXI)*.

Capítulo 3

Control de concurrencia en un sistema de autoría colaborativa

En capítulos anteriores se presentó un panorama general de los sistema distribuidos y los sistemas groupware, principalmente se hizo referencia a los temas de mecanismos de control de concurrencia y sistemas de autoría, los cuales han sido de gran apoyo para conocer algunos de los procesos de la escritura colaborativa. Ahora, corresponde describir la propuesta de este trabajo de tesis.

Actualmente se está creando un sistema de autoría llamado *Editor coLaborativo de documentos XML en Internet, y se* hará referencia a él mediante las siglas *ELXI*. El objetivo de este sistema es la elaboración de documentos en forma concurrente, apoyada en algunas fases de edición y en herramientas groupware. Su uso está dirigido a la comunidad científica para la elaboración de artículos, sin embargo también puede ser utilizado por aquellos sectores públicos o privados cuyo interés principal es la elaboración de documentos (normas, revistas, periódicos, tesis, libros, etc.)

El diseño de ELXI se ha basado en algunas desventajas detectadas en otros editores. Por ejemplo, de los sistemas de autoría descritos en el capítulo dos, se observó que no todos permiten la edición concurrente, la cual es importante para aquellos grupos de trabajo en donde existe más de un escritor. Así como, la falta de prevención y solución de conflictos de concurrencia, siendo el caso de PREP. Otra característica que no se encontró en los sistemas de autoría mencionados (a excepción de aquellos que trabajan sobre la Web), es la portabilidad, ya que solamente pueden ser ejecutados en determinadas plataformas (Unix o MacOS).

Aunque ELXI continua en la fase de desarrollo, su arquitectura ha sido definida y puesta en marcha, permitiendo la elaboración de documentos. En este momento, la prioridad del proyecto ELXI es almacenar y administrar los documentos, así como mantener la consistencia de la información de los mismos. Para lograrlo, en este capítulo se propone un mecanismo de control de concurrencia, el cual será implementado de acuerdo al análisis realizado sobre las necesidades del sistema actual.

A continuación se describen las características y el funcionamiento que conforman el estado inicial en que se adquiere el sistema ELXI. Posteriormente, se definirá el problema de concurrencia y su posible solución. En seguida, se incorpora el mecanismo de control de concurrencia siguiendo las etapas de análisis, diseño e implementación, en las cuales se utiliza la terminología del Lenguaje Unificado de Modelado para su explicación.

3.1 Sistema de autoría colaborativa ELXI

Debido a que ELXI ha sido desarrollado en el lenguaje de programación JAVA, incorporarse a este proyecto no ha sido una tarea difícil. Debido a su naturaleza, ELXI está siendo programado de manera modular, esta característica hace posible que un grupo de desarrolladores puedan trabajar en diferentes partes del proyecto. Otra ventaja que ha ofrecido JAVA es la reutilización del código, lo cual ha permitido enfocarse más en el verdadero problema. Asimismo, concede la facilidad de ejecutar la aplicación final en diferentes plataformas. Otra característica de gran interés es la distribución de objetos, que mediante la invocación de métodos remotos es posible acceder a los recursos de diferentes computadoras.
Actualmente, la arquitectura del sistema ELXI y algunos procesos como la administración y elaboración de documentos, creación de sesiones de trabajo y envíos de mensajes, han sido implementados. Sin embargo, el proyecto ELXI continua en la fase de desarrollo. Uno de los requerimientos que se demanda es la existencia de un control de concurrencia en la edición de documentos. Con el fin de proponer este mecanismo, primero se describirá el estado inicial del que se parte, es decir cómo está integrado y cómo trabaja ELXI, de tal forma que se conocerán los procesos creados y también cuales serán afectados.

3.1.1 Arquitectura ELXI

La arquitectura de ELXI mostrada en la figura 3.1, es un sistema híbrido de tres capas: cliente ELXI, servidor ELXI y servidor de base de datos.
El servidor ELXI es el encargado de controlar todas las acciones de los colaboradores sobre los documentos y está implementado mediante la Invocación del Método Remoto (RMI[32]) en JAVA. Esto le permite al servidor crear objetos que serán accedidos remotamente por los clientes mediante referencias llamadas *interfaces*. En el diagrama de distribución de procesos mostrado en la figura 3.2, se observa que sólo hay un proceso servidor, el cual incorpora diferentes mecanismos como el de conexión, concurrencia, replicación y administración de documentos.

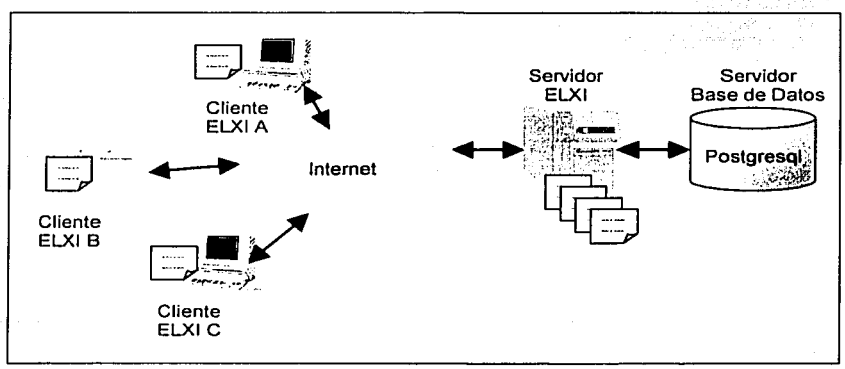


Figura 3.1. Arquitectura de ELXI.

[32] Del inglés *"Remote Method Invocation"*.

Una de las notables aportaciones de ELXI es el almacenamiento de la información en documentos estructurados mediante el Lenguaje de Marcado Extensible (XML[33]), de esta forma, su manejo es como el de un objeto, permitiendo que el acceso y el almacenamiento de datos se realice de forma dinámica. Los documentos son archivos con extensión xml que residen en el sistema de archivos donde se ejecuta el proceso servidor que los administra. En la figura 3.2 se observa que la relación entre el proceso servidor y el sistema de archivos es uno a uno. Esta relación se establece debido a que el proceso servidor se comunica con un sólo sistema de archivos para la localización, lectura y escritura de documentos.

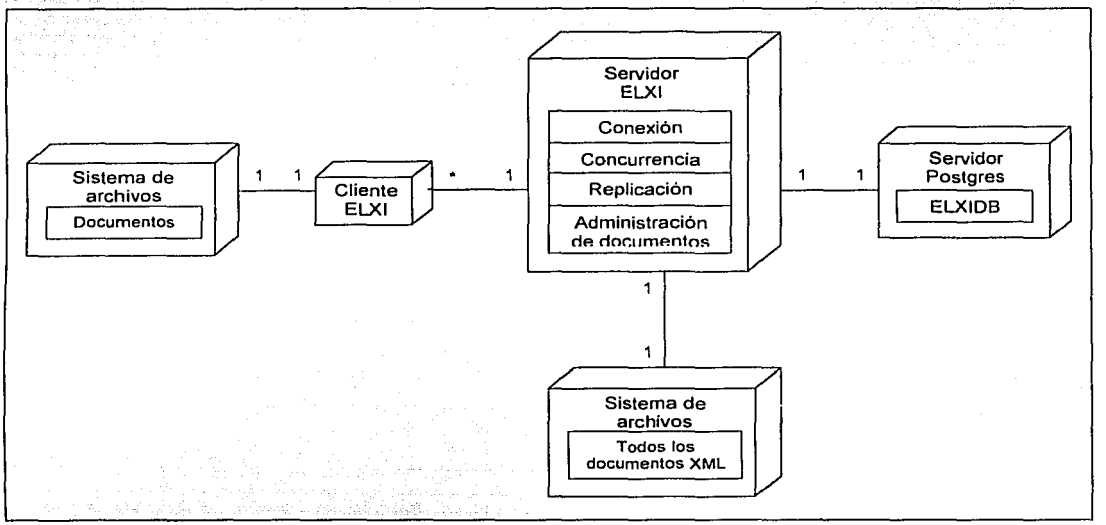


Figura 3.2. Diagrama de distribución.

Otro proceso en ejecución es el servidor Postgres, el cual es un manejador de bases de datos que establece una relación uno a uno con el servidor ELXI. Todas las peticiones que envía el servidor ELXI al servidor Postgres, se ejecutan sobre la base de datos ELXIDB y son manejadas mediante la API de JAVA, conocida como JDBC[34].

El propósito de la base de datos ELXIDB es el de almacenar la información concerniente a cada usuario y documento registrado en ELXI, de tal forma que sea utilizada para coordinar las actividades de los colaboradores y también para fomentar la conciencia en grupo. En la figura 3.3 se muestra el diagrama entidad relación de la base de datos ELXIDB. Existen cuatro tablas nombradas como *usuarios*, *archivos*, *usuarios_conectados* y *usuario_archivo*. Esta última tabla rompe la relación muchos a muchos entre las tablas *usuarios* y *archivos*, debido a que un usuario puede acceder a varios archivos y un archivo puede ser accedido por varios usuarios. La tabla *usuarios* almacena información de los usuarios de ELXI, así como también su contraseña[35] para ingresar al sistema. Cada documento esta identificado por un número único dentro de la tabla

[33] Del término en inglés *"eXtensible Market Language"*.
[34] Del término en inglés *"Java Data Base Connectivity"*.
[35] La contraseña no es almacenada como un texto en claro, sino es un compendio de mensaje creado mediante MD5.

archivos, en la cual también se encuentra la información respecto al nombre del archivo, cuando fue creado y por quién.

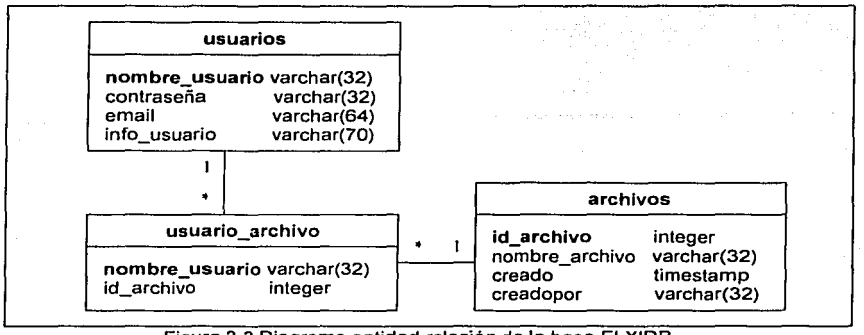


Figura 3.3 Diagrama entidad relación de la base ELXIDB.

Continuando con el diagrama de distribución de la figura 3.2, el proceso cliente es el encargado de atender las peticiones del usuario final, así como también direccionarlas al servidor ELXI cuando se requiera. La relación que guarda el proceso servidor con respecto al proceso cliente es de uno a muchos. Entonces el proceso servidor para atender las solicitudes de los diferentes clientes, mantiene un puerto abierto para escuchar las llamadas a los métodos de los objetos remotos.
El proceso cliente también se comunica con el sistema de archivos de la máquina en la que se esta ejecutando, ya que en él se almacenan únicamente las réplicas de los documentos que el usuario final ha solicitado. Este almacenamiento permite que el usuario final pueda trabajar en modo sin conexión[36].

3.1.2 Administración de documentos

Los documentos XML son almacenados en un directorio específico nombrado *datos,* que se encuentra localizado dentro del directorio donde reside el programa de ejecución del proceso servidor (ver figura 3.4).

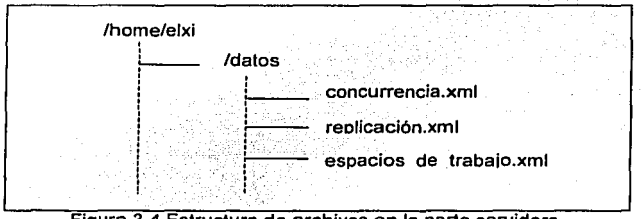


Figura 3.4 Estructura de archivos en la parte servidora.

En la parte cliente se maneja una estructura de archivos similar. La diferencia radica en que se define un directorio de usuario con el nombre del mismo, el cual contendrá los archivos que han

[36] No es necesario que el cliente envíe al servidor una solicitud de *obtener_documento* para poder visualizar el documento, ya que éste se encuentra localmente en la máquina del usuario.

solicitado. Dentro de este directorio, también se crea un archivo de información de usuario[37], estructurado mediante XML, que contiene datos como el nombre de usuario, contraseña y el nombre de los documentos con los que ha trabajado. En la figura 3.5 se muestra el directorio de datos de ELXI, dentro de este existe otro con el nombre del usuario silvia, el cual contiene los documentos *concurrencia.xml* y *espacios_de_trabajo.xml* y el archivo de información *silvia.xml*.

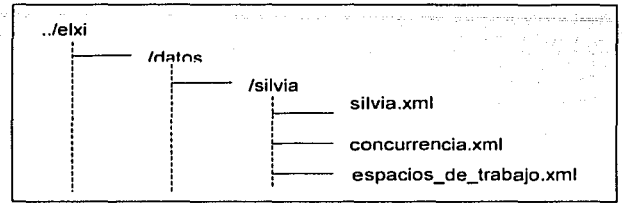


Figura 3.5. Estructura de archivos en el cliente.

Todos los archivos con extensión xml son documentos estructurados mediante el Lenguaje de Marcado Extensible. Éstos tienen las características de ser ***bien construido***, por que respetan la estructura y sintaxis definida por la especificación de XML, y ***válidos***, debido a que cumplen las reglas de la Definición del Tipo de Documento (DTD[38]). La tarea de la DTD es establecer el modelo de contenido de los documentos y los tipos de datos. Esto contribuye a que las entidades del documento sean localizadas fácilmente. En la figura 3.6 se muestra un ejemplo de la estructura de un archivo de tipo documento y de información de usuario con sus respectivas DTD.

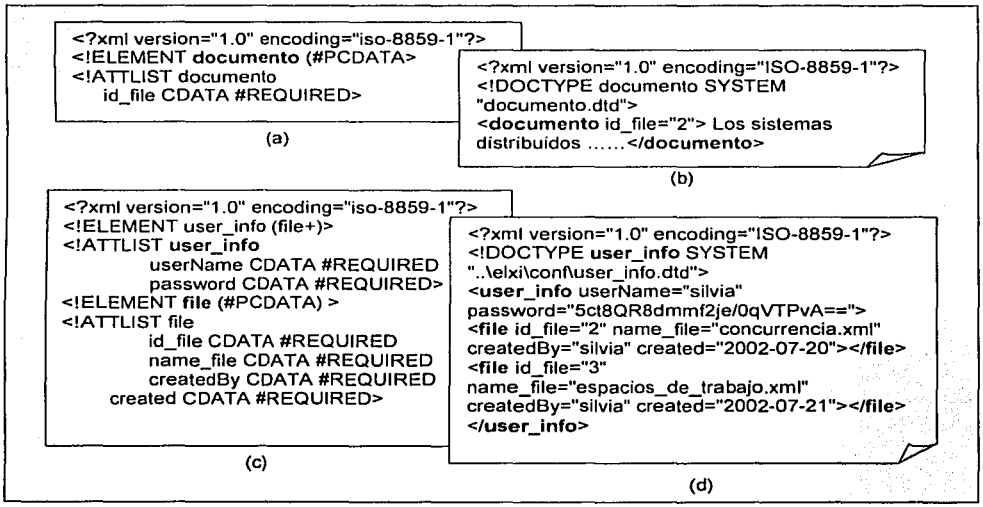


Figura 3.6. (a) DTD de documento. (b) Archivo de tipo *documento* que contiene el texto. (c) DTD de información de usuario. (d) Archivo de tipo *información de usuario* y contenido.

[37] Esta información es utilizada por el proceso cliente en el modo sin conexión.
[38] Del inglés *"Document Type Definition"*.

El esquema de la DTD mostrado en la figura 3.6.a, es usado para validar los documentos. En él, se definen únicamente el elemento *documento* y su atributo *id_file*, el cual indica el número de identificación del documento registrado en la base de datos. En la figura 3.6.b se observa el documento estructurado y entre las etiquetas *<documento id_file="2">* y *</documento>* está el contenido. La DTD de la figura 3.6.c es utilizada para validar la información de usuario y define los elementos *user_info* y *file*, cada uno con sus atributos respectivos. La información del usuario es almacenada en el documento estructurado de la figura 3.6.d. La etiqueta *user_info* hace referencia a la contraseña de un usuario mediante el atributo *password* y la etiqueta *file* localiza el identificador del documento (*id_file*), su nombre (*name_file*), quien lo creo (*createdBy*) y cuando (*created*). Este tema es abordado con más detalle en [RoAn].

3.1.2 Replicación en ELXI

El mecanismo de replicación consiste en enviar, desde la parte servidora, las actualizaciones de los documentos modificados a los procesos clientes activos. Para poder llevarlo a cabo, cada vez que un cliente se conecta al proceso servidor, éste almacena su interfaz[39] en memoria. De tal manera que, cuando se requiera actualizar el documento de un colaborador, se utiliza la interfaz del proceso cliente asociada al colaborador correspondiente para enviarle la copia (réplica) del documento original.

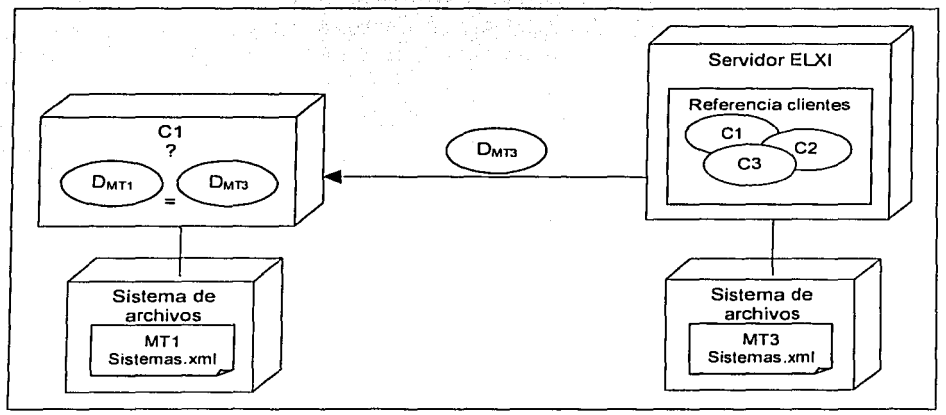


Figura 3.7. Replicación de documento.

Otra cuestión que interviene en las actualizaciones del documento es saber si realmente se tiene que actualizar el documento. Para ello, se utilizan marcas de tiempo asociadas al documento original, así como también en las réplicas. De esta forma, cuando una réplica llega es comparada con la copia que reside en la parte cliente y si la marca de la copia local es menor entonces es actualizada. En la figura 3.7, el servidor ELXI envía un objeto D con la marca de tiempo MT3 al

[39] Define el conjunto de métodos o acciones pertenecientes a un objeto remoto que pueden ser solicitadas por otros objetos.

cliente C1. El proceso cliente compara el objeto llegado con el objeto D que hace referencia al documento local con marca de tiempo MT1. Como la marca de tiempo MT3 es mayor a MT1, entonces el documento es actualizado (sobrescrito).
En este mecanismo de replicación también se manejan el envío de mensajes de colaboradores y mensajes de estado del sistema, pero únicamente funciona para los procesos clientes activos. Este tema es abordado con más detalle en [MuCe].

3.1.2 Interacción de objetos en ELXI

El sistema ELXI esta diseñado mediante el Lenguaje Unificado de Modelado. En sus primeras fases de desarrollo se identificaron principalmente tres actores que interactúan con el sistema: el colaborador, el sistema de archivos y el manejador de base de datos Postgres (ver figura 3.8).

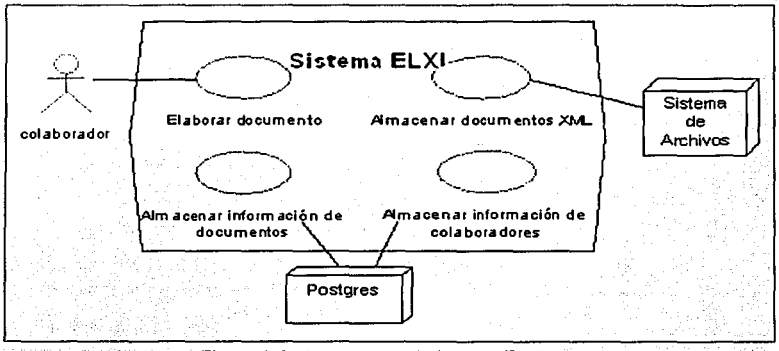


Figura 3.8. Actores en el sistema ELXI

El colaborador es un usuario del sistema ELXI que tiene por objetivo participar en la elaboración de un documento. Esta participación se puede dar mediante las acciones de escritura y lectura. Como anteriormente se mencionó, el sistema de archivos realiza el almacenamiento de los documentos en un directorio en específico. Mientras que el servidor de bases de datos se encargará de realizar las peticiones del proceso servidor sobre la base de datos ELXIDB, para controlar la información de los colaboradores y los documentos.
En el transcurso del desarrollo y análisis de ELXI se crearon las clases necesarias para tener un sistema funcional. En la figura 3.9 se muestran algunos paquetes que agrupan las diferentes clases. Se observa que tanto el paquete *cliente* como el paquete *servidor* importan clases del paquete *xml*, el cual contiene clases referentes al manejo de documentos. Otro paquete que utiliza el proceso *cliente* para la creación del ambiente gráfico es *swing*. Éste también es requerido por el paquete *editor*, el cual integra las clases que generan la interfaz del editor. Asimismo, se auxilia del paquete *w3c* para el Modelo de Objeto de Documento (DOM[40]), el cual permite acceder y manipular un documento XML. Por último, el paquete *util* contiene clases que intervienen en la ejecución de métodos del proceso cliente y servidor.

[40] Del inglés *"Document Object Model"*.

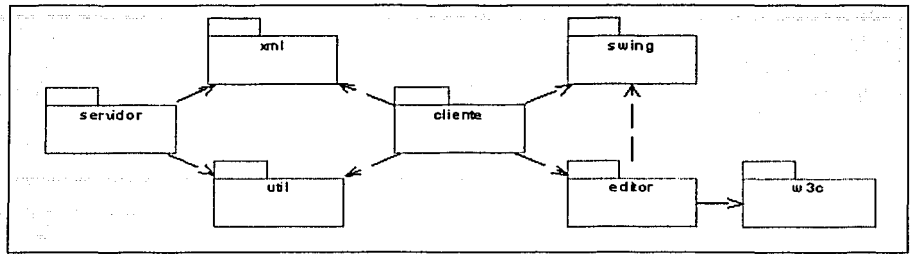


Figura 3.9. Algunos paquetes creados en ELXI para la agrupación de clases.

A continuación se explica el funcionamiento de algunas clases que integran los paquetes anteriormente mencionados (ver figura 3.10).

Para implementar el servidor ELXI se diseñó la clase *ServidorElxiImpl*, mediante la cual se genera un objeto que es exportado y registrado en un puerto TCP para que pueda ser invocado remotamente por los objetos cliente. La referencia que realiza el objeto cliente al objeto servidor no es directa, sino se lleva acabo a través de una interfaz remota implementada por la clase *ServidorElxiImpl* y nombrada *ServidorElxi* , la cual identifica al objeto susceptible de ser accedido.

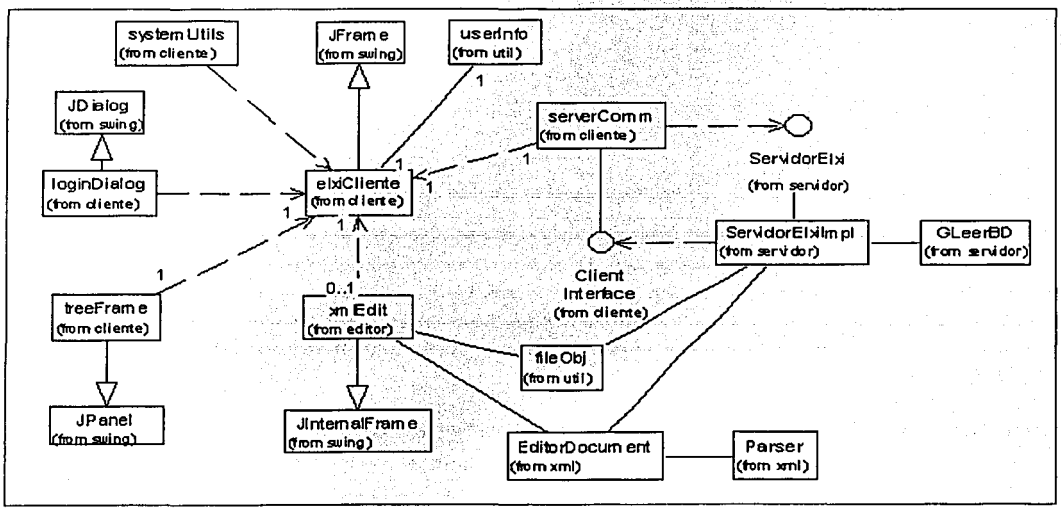


Figura 3.10. Diagramas de clases del sistema ELXI.

Otra clase en el paquete *servidor* es *GLeerBD*, utilizada para establecer una comunicación con el servidor de Bases de Datos. De igual manera, las clases *EditorDocument* y *fileObj* son utilizadas por la clase *ServidorElxiImpl* para la obtención de archivos en el sistema de archivos y el manejo de la información. A su vez, se recurre a la clase *Parser*, en la cual se implementan métodos encargados de analizar sintácticamente el contenido de un archivo para verificar que sea un documento XML válido.

En el paquete *cliente* se encuentra la clase *elxiCliente*, la cual es fundamental para generar la interfaz del espacio de trabajo. Otra clase es *treeFrame*, empleada para crear un panel que lista los nombres de los documentos. Este panel ha sido nombrado como árbol de documentos y forma parte del espacio de trabajo del usuario. La clase *userInfo* es utilizada para almacenar la información del usuario que esta en sesión. Asimismo, se recurre a la clase *xmlEdit* para construir la ventana del editor. Dentro del espacio de trabajo se pueden crear varios objetos de tipo *xmlEdit*, los cuales muestran un documento único en una ventana de edición.
Finalmente, la clase *serverComm* incorpora los métodos necesarios para establecer la comunicación con el servidor. Esta clase también implementa una interfaz llamada *ClientInterface*, la cual permite que el servidor se pueda comunicar con el cliente cada vez que exista un cambio de estado en los documentos. Este modo de comunicación fue desarrollado mediante el modelo de los *callbacks*, en el cual se define el método *UnicastRemoteObject.exportObject()* que al ser ejecutado, exporta el objeto *ClientInterface* en el puerto donde residen los objetos remotos.

Sesión en ELXI

Una sesión de ELXI define un espacio de trabajo único para cada colaborador y sólo la podrá iniciar aquel que ya haya sido registrado en el sistema. El administrador, o bien, un colaborador que ya forme parte de ELXI, pueden registrar a otros colaboradores en el sistema.

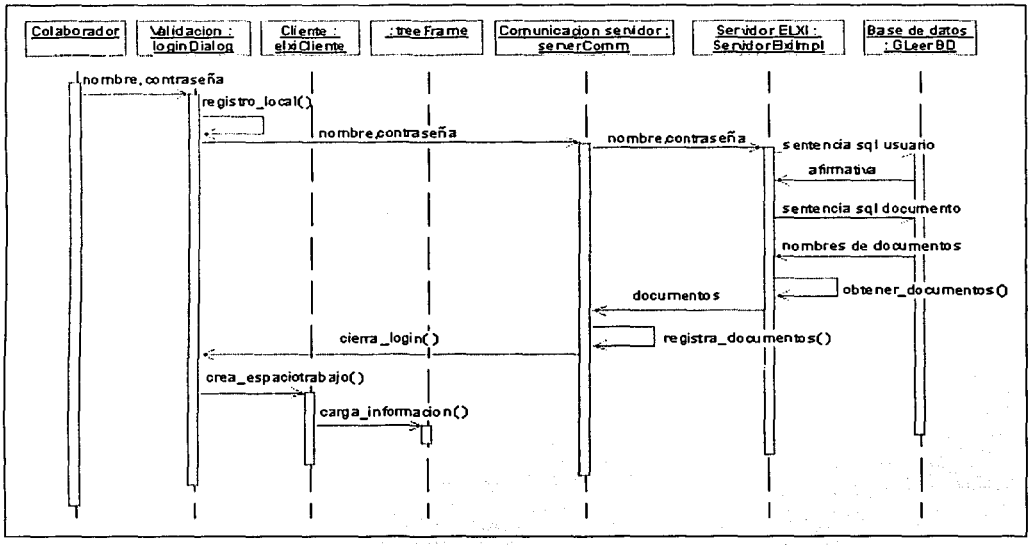


Figura 3.11. Diagrama de secuencia "Validación remota".

Para iniciar una sesión, el colaborador debe ejecutar el proceso cliente en su máquina local. Enseguida se despliega una ventana de validación que solicita un nombre y una contraseña. Posteriormente el proceso cliente verifica si el colaborador ha entrado anteriormente al sistema desde esa máquina (es decir, busca si existe una estructura de archivos ELXI para ese

colaborador). En caso contrario, se ejecuta un mecanismo de conexión que consiste en enviar el nombre y la contraseña del colaborador al proceso servidor, para que sea validado y registrado como un usuario activo. La validación consiste en comprobar que el nombre y la contraseña correspondan a las almacenadas en la tabla *usuarios* de la base de datos *ELXIDB*. Si la operación fue exitosa, el *servidor ELXI* realiza una búsqueda de información sobre la base de datos para obtener el nombre de los documentos con el fin de localizarlos en el sistema de archivos. Después de que el proceso servidor obtiene los documentos, los envía al proceso cliente quien se encargará de guardarlos en el sistema de archivos local y además desplegará sus nombres en el árbol de documentos del espacio de trabajo. La figura 3.11 muestra el diagrama de secuencia para este escenario.

Por otra parte, si el colaborador ya había entrado al sistema ELXI desde su máquina, la validación la realiza el proceso cliente, utilizando el archivo de información de usuario almacenado en su estructura de archivos. Si el proceso de validación fue exitoso, el cliente mostrará el espacio de trabajo, del cual, el colaborador podrá seleccionar un documento desde el árbol de documentos (ver figura 3.12).

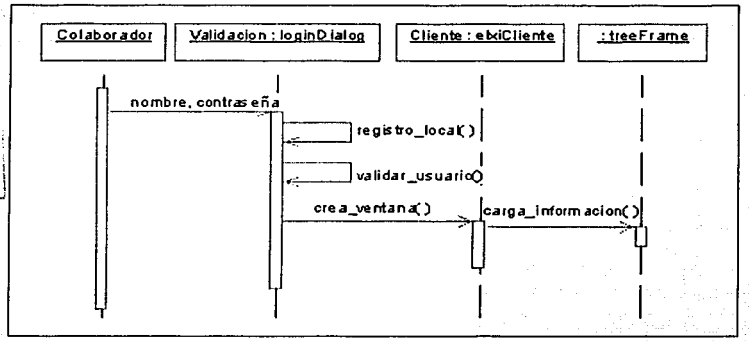


Figura 3.12.Diagrama de secuencia "Validación local de usuario".

Un colaborador puede obtener un documento al seleccionar la opción *abrir_documento* en el espacio de trabajo. La operación que realiza el proceso cliente es la búsqueda física del documento en el sistema de archivos a través de la clase *EditorDocument*. Después se crea el

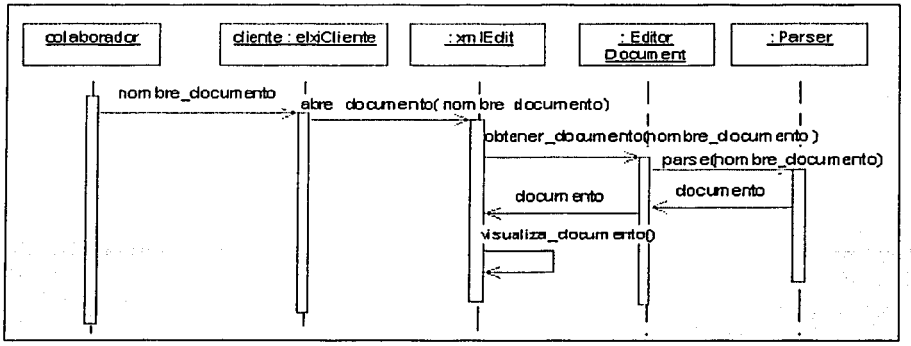


Figura 3.13. Diagrama de secuencia "Obtención de documento".

objeto de tipo *Parser*, el cual se encargará de realizar un análisis sintáctico para verificar que se trata de un documento xml válido. Si es el caso, se construye una estructura de árbol del documento, de tal manera que cada entidad (identificada por la etiqueta *documento*), es considerada como un objeto nodo. Posteriormente, el contenido de cada nodo es desplegado en la ventana del editor. El diagrama de secuencia se muestra en la figura 3.13.

Para crear un documento, el colaborador debe proporcionar el nombre del mismo. Posteriormente, el proceso cliente envía al servidor la solicitud *crear_documento*, junto con el objeto de tipo *fileObject* que contiene atributos referentes al nombre del colaborador y al nombre del documento. Esta información es almacenada por el servidor en la tabla *archivos* de la base de datos, la cual regresa un identificador único para ese documento. Después, el servidor crea el documento en el sistema de archivos, anexándole su identificador. Una vez creado el documento físicamente, el servidor devuelve al proceso cliente el identificador del documento para que sea creado en el sistema de archivos de la máquina cliente. Realizada esta operación, el proceso cliente hace visible el nuevo documento al usuario final. Por otra parte, el servidor se encarga de replicar el documento a los demás procesos cliente activos. En el diagrama de secuencia de la figura 3.14 se muestran las operaciones que involucran la creación de un documento.

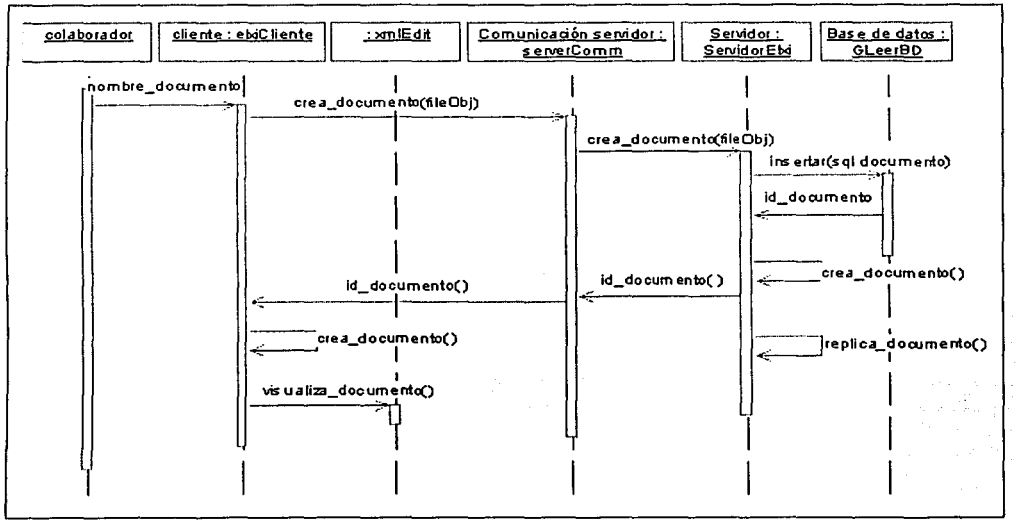


Figura 3.14. Diagrama de secuencia "Creación de un documento".

Una vez que un colaborador decide guardar sus modificaciones en el documento, el proceso cliente envía al servidor la solicitud *guardar_documento* junto con el objeto de tipo *fileObj*, que contiene el nombre de usuario, el identificador del documento y el texto. El servidor obtiene el nombre del documento a través del identificador de documento registrado en la base de datos, con el propósito de buscarlo en el sistema de archivos. Una vez localizado el archivo, mediante los objetos de tipo *EditorDocumento* y *Parser* se sobrescribe el contenido del documento. Posteriormente, se envía una respuesta afirmativa al proceso cliente indicando que se realizó el

guardado del documento. Enseguida, el proceso cliente almacena el contenido del documento en el archivo de la máquina local (ver figura 3.14).

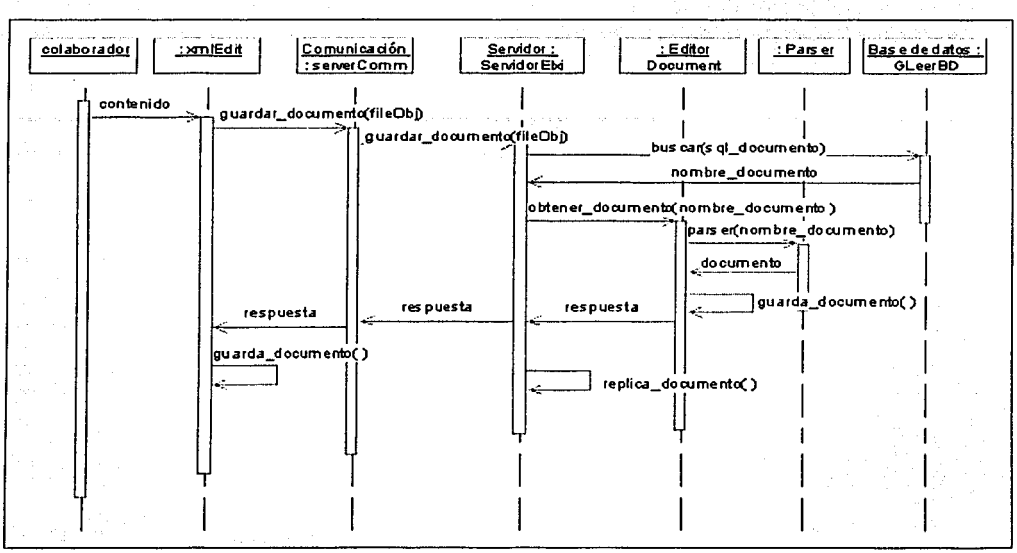


Figura 3.15. Diagrama de secuencia "Guardar cambios en un documento".

En esta sección se describieron principalmente los casos de ingreso a una sesión y manejo de documentos en ELXI, debido a que estos procesos constituyen la base fundamental para proponer el mecanismo de control de concurrencia desarrollado en las siguientes secciones.

3.2 Concurrencia en la edición de documentos

El principal objetivo de ELXI es ofrecer una aplicación que permita la elaboración de documentos de manera colaborativa. Para llevarlo a cabo, uno de los mecanismo que se implementó fue el de la replicación, con la finalidad de que cada autor tuviera una copia del documento con la cual trabajar. Sin embargo, con este mecanismo surgen nuevas problemáticas, principalmente la inconsistencia de la información de los documentos, en el momento en que los autores realizan modificaciones en la misma parte del documento. Cabe mencionar que las propuestas que se desarrollan más adelante se implementarán sobre la arquitectura híbrida en la que trabaja ELXI (ver sección 3.1.1 de este capítulo).

3.2.1 Definición del problema de concurrencia

El esquema inicial del que se parte es el siguiente. En ELXI, cada colaborador escribe sobre una copia del documento original, la cual reside en la máquina cliente. Ahora bien, si un grupo de colaboradores decide trabajar sobre un mismo documento, entonces la edición se realiza sobre la

copia y por lo tanto cada quien tiene una versión diferente. Si un colaborador decide realizar sus cambios de manera permanente en el documento original, se requiere que el proceso cliente solicite al servidor ejecutar la operación de guardar. En esta solicitud, el proceso cliente tiene que enviar la copia del documento en donde el colaborador realizó sus aportaciones. Enseguida, el proceso servidor se encarga de localizar el documento XML original en el sistema de archivos, para sobrescribir en él la copia enviada por el cliente. Después, el mismo proceso servidor se encargará de replicar las nuevas aportaciones del documento a los demás clientes activos. Asimismo, si estos clientes tiene una copia del documento a actualizar, entonces serán sobrescritas. Bajo esta situación, se puede dar el caso de que un grupo de colaboradores decida guardar al mismo tiempo los cambios realizados en cada una de sus copias, entonces la primera solicitud que llegue al servidor será ejecuta y posteriormente las demás en un orden consecutivo. Sin embargo, la segunda solicitud al ejecutarse sobrescribirá el documento original, por lo que los cambios del primer autor se perderán. Y si existe un tercer colaborador, los cambios del segundo también se perderán. El documento quedará inconsistente para todos lo autores que guardaron sus cambios, a excepción del último. Así, cuando un colaborador desee obtener el documento, solo verá las aportaciones de la última actualización. En este caso, se detecta una pérdida de datos por lo que es necesario implementar un control de concurrencia en la edición que será expuesto en la siguiente sección.

3.2.2 Análisis

Una posible solución al problema expuesto en la sección anterior, es hacer exclusivo el documento para un solo colaborador, de tal forma que al concederle la acción de escritura ningún otro pueda editarlo en ese momento. Sin embargo, con esta solución se perdería la concurrencia en la edición.
Haciendo referencia al manejo de la concurrencia de algunos editores colaborativos como ALLIANCE, MACE y DUPLEX, en este trabajo de tesis se han propuesto tres esquemas para resolver el problema citado:

1. Estructura del documento, en el cual se defina la unidad mínima de concurrencia.
2. Asignación de responsabilidades a los colaboradores del sistema ELXI.
3. Control del proceso de escritura sobre la unidad de concurrencia.

Con respecto al primer esquema se ha considerado el siguiente caso. Los colaboradores pueden estar trabajando en diferentes partes del documento, entonces cada una de esas partes pueden ser guardadas y actualizadas sin interrumpir el trabajo de cada colaborador. Esta forma de trabajo fomenta la idea de dividir el documento en segmentos. Sin embargo, surge la cuestión de la definición del tamaño del segmento, lo cual influye en el grado de concurrencia (número de autores escribiendo al mismo tiempo) y la granularidad de edición (tamaño del segmento). Estos dos parámetros guardan una relación inversamente proporcional, ya que si se requiere que el índice de concurrencia sea grande la granularidad de edición será pequeña, es decir, si el documento es dividido en pequeños segmentos existirá un número grande de colaboradores que trabajen en cada uno de los segmentos. Pero si la granularidad es grande el grado de concurrencia se reduce, por ejemplo si el documento es dividido en dos o tres segmentos grandes, el número de colaboradores editando concurrentemente se reduce.

Otro factor que interviene para definir el tamaño del segmento es la forma de organización de los colaboradores. Es posible que para determinado documento se requiere la participación de varios escritores , o bien, sólo la intervención de unos cuantos. Entonces el grado de concurrencia y la granularidad de un documento estarán determinados por la forma en que los colaboradores se organizan para la división del trabajo de edición.

Con este panorama, se ha establecido que en el sistema ELXI los propios colaboradores definan los parámetros de concurrencia y granularidad. Por lo tanto, se ha propuesto que el documento sea dividido en unidades llamadas fragmentos, las cuales pueden ser interpretados por el colaborador como líneas, párrafos, secciones, capítulos, etc.

Una vez definida la estructura del documento, un conjunto de colaboradores puede trabajar concurrentemente sobre él, pero existe la posibilidad de que dos colaboradores editen al mismo tiempo el mismo fragmento. Para este problema se ha recurrido al segundo esquema que es la asignación de responsabilidades a los usuarios del sistema. En él, se propone trabajar con permisos de acceso, pero antes de definirlos se realizará un análisis de las acciones que los colaboradores pueden realizar sobre los documentos.

En ELXI todos los colaboradores tienen acceso a todos los documentos, permitiendo que cualquier colaborador pueda escribir aunque no tenga el interés o el compromiso de trabajar en el documento. Esta problemática condujo a la implementación de grupos de trabajo [SiKl97], a través de los cuales se invita a participar a colaboradores que se desea que estén involucrados en el proyecto de edición. De esta manera, los documentos que se desarrollan en un grupo de trabajo sólo podrán ser obtenidos por los miembros de ese grupo, de esta forma se limita el acceso a otros colaboradores de otros grupos. Asimismo, existe la posibilidad de que los colaboradores que trabajan en un documento no deseen que sea visualizado por otros miembros del mismo grupo de trabajo, por lo que también hay una restricción de acceso a los miembros en la obtención del documento. Entonces, si existe un conjunto de colaboradores C_i y algunos de ellos o todos son miembros de un grupo G_i, pueden o no obtener el documento D_i según sus permisos de acceso. En la figura 3.16, se observa que el colaborador C_1 miembro del grupo G_1, no puede obtener el documento D_1 aunque pertenezcan al mismo grupo. Por el contrario, los colaboradores C_2 y C_3 pueden obtenerlo. Por otra parte, el colaborador C_4 puede acceder a los documentos D_2 y D_3 pertenecientes a diferentes grupos, debido a que es miembro de los grupos G_2 y G_3, y además tiene ese permiso. El colaborador C_6 también tiene acceso a dos documentos que pertenecen al grupo G_3, del cual es miembro.

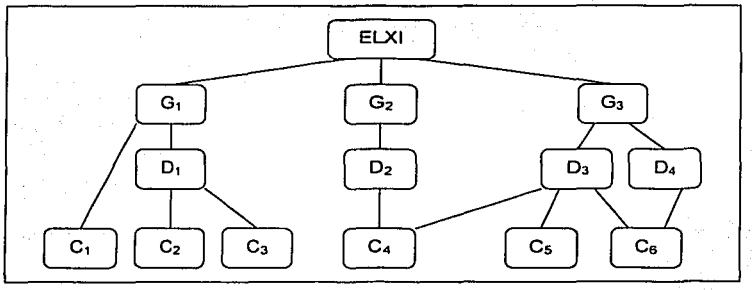


Figura 3.16. Estructura de grupo jerárquica.

De esta manera, un colaborador puede acceder a un documento siempre y cuando sea miembro del grupo al que pertenece el documento y además cuente con el permiso correspondiente para visualizarlo o editarlo. Para definir los permisos que tiene un colaborador sobre un documento se hará referencia a los roles que puede adquirir.
Los roles son asignados de acuerdo al tipo de trabajo que realizarán los colaboradores en el documento y en los fragmentos. Para el documento, se han propuesto dos roles:

1. **Administrador de documento**. Colaborador que asigna roles de documento a otros colaboradores.
2. **Autor de documento**. Colaborador que puede obtener el documento para trabajar en él.

Los roles tienen asignados permisos y para el *administrador de documento* son los de otorgar roles a otros colaboradores, obtener el documento para leer o escribir y agregar fragmentos al documento. Para el rol *autor de documento*, sólo se puede adquirir uno de los siguientes tres permisos: obtener el documento, agregar fragmentos al documento o ninguno de los anteriores, es decir, el autor se le niega la obtención del documento (sin permisos). La asignación de permisos para el administrador y autor de documento también se pueden observar en la figura 3.17.

Rol	Otorgar roles	Obtener documento	Agregar fragmentos	Sin permisos
Administrador	√	√	√	
Autor		(√)[41]	(√)	(√)

Figura 3.17.Tabla de roles de colaboradores en el documento.

La manera de permitir que se escriba en el documento dependerá de los permisos que el autor de documento tenga sobre el fragmento, por lo que surgen nuevos roles pero ahora referentes al fragmento.

1. **Administrador de fragmento**. Autor que asigna roles de fragmento.
2. **Escritor de fragmento**. Autor que puede escribir en un fragmento.
3. **Lector de fragmento**. Autor que sólo le es permitido leer el fragmento.

Los permisos que tiene el *administrador de fragmento* son el de otorgar roles, escribir y leer el documento. Para el *escritor de fragmento* los permisos son el de escribir y leer el fragmento. Por último, el *lector de fragmento* que tan sólo se le permite leer. La asignación de permisos para el administrador y autor de documento también se pueden observar en la figura 3.18.

Rol	Otorgar roles	Escritura	Lectura
Administrador	√	√	√
Escritor		√	√
Lector			√

Figura 3.18. Tabla de roles de autores en fragmentos.

Con este esquema de responsabilidades mediante roles y permisos se permite o se niega la operación de escritura en un fragmento. Sin embargo, aún persiste el problema de que dos autores escriban en el mismo fragmento al mismo tiempo. Para resolverlo se recurrirá al tercer esquema que es el control del proceso de escritura sobre la unidad de concurrencia, el cual será

[41] El paréntesis representa que el permiso puede ser opcional

implementado por alguno de los tres mecanismos de concurrencia presentados en el capítulo uno. A continuación se realiza un análisis para saber cual de ellos es el óptimo para el sistema ELXI.

El mecanismo de control de concurrencia optimista se basa en la baja probabilidad que tienen dos transacciones cliente de acceder al mismo tiempo el mismo objeto, la cual es una desventaja en ELXI ya que los fragmentos serán concurridos por diferentes colaboradores.

Por otra parte, el mecanismo de ordenamiento de marcas de tiempo es preferible en sistemas donde predominan operaciones de lectura. Para ELXI, el problema radica en la escritura concurrente.

El bloqueo parece ser el mejor mecanismo para manejar la concurrencia en archivos de texto, de tal manera que se controle el acceso de cada fragmento para que diferentes transacciones concurrentes puedan realizar la acción de lectura y sólo una transacción realice la de escritura. Una de las desventajas de este mecanismo es el bloqueo mutuo, sin embargo hay que considerar que la escritura en ELXI es totalmente asíncrona, es decir, las aportaciones al documento son permanentes cuando el autor decide guardarlas en el documento original que permanece en el servidor. Por lo tanto, se puede controlar y evitar el bloqueo mutuo, garantizando que el proceso de escritura en un fragmento sólo lo puede hacer un autor a la vez. De igual manera, se propone que el bloqueo dure sólo un intervalo de tiempo para cada colaborador. Sin embargo, determinar el tiempo de bloqueo no es trivial, ya que si se establece un periodo como fijo, es posible que el autor requiera más tiempo en la edición, o bien que sólo necesite algunos segundos. Es por ello que se propone que el mismo autor defina este tiempo, sin la restricción de que pueda solicitar el bloqueo varias veces en un día.

A continuación se realizará el diseño del mecanismo de control de concurrencia expuesto en esta sección, respetando la arquitectura de ELXI y el propósito de la base de datos, sin embargo, es posible que algunos de los mecanismos ya implementados sean modificados.

3.2.3 Diseño

Para organizar los múltiples comportamientos que surjan de los esquemas que se propusieron en el análisis anterior, se recurrirá a definir cuáles son los procesos o actividades a modelar mediante el diseño de diagramas de caso de usos, escenarios y diagramas de secuencia.

Debido a que la fase inicial de la edición es la planeación, es conveniente definir primero la parte de la organización de trabajo mediante la creación de grupos, posteriormente la estructura del documento en fragmentos, así como el proceso de asignación de roles de documentos y fragmentos. Finalmente, el proceso de edición mediante el mecanismo de bloqueo.

Suponiendo que en ELXI existiera un número determinado de colaboradores, de los cuales, alguno está interesado en convocar a otros para elaborar un documento. Por lo tanto, este colaborador debe crear su propio grupo de trabajo. Esta actividad es motivo de crear el caso de uso de la figura 3.19.

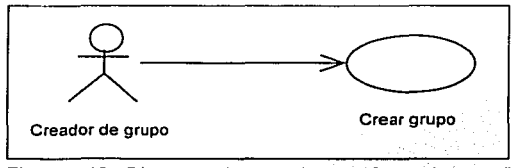


Figura 3.19 . Diagrama de caso de uso "Crear de grupo"

El diagrama de secuencia para la creación de un grupo se muestra en la figura 3.20. Primero se requiere que el colaborador proporcione el nombre del grupo. Posteriormente, el proceso cliente enviará la solicitud *crear_grupo* al servidor, quien se encargará de registrar el nombre del grupo en la base de datos. Enseguida, creará en el sistema de archivos un directorio con el nombre del grupo, donde se almacenarán todos los documentos XML creados por los miembros de ese grupo. Si esta operación fue un éxito, en la parte cliente se creará también un directorio perteneciente al colaborador y en el panel *treeFrame* del escritorio de trabajo se visualizará una carpeta con el nombre del grupo. Pero en el caso de que el proceso servidor no pueda crear el directorio, el proceso cliente visualizará un mensaje de operación no ejecutada. Asimismo, lo realizado en la base de datos será anulado.

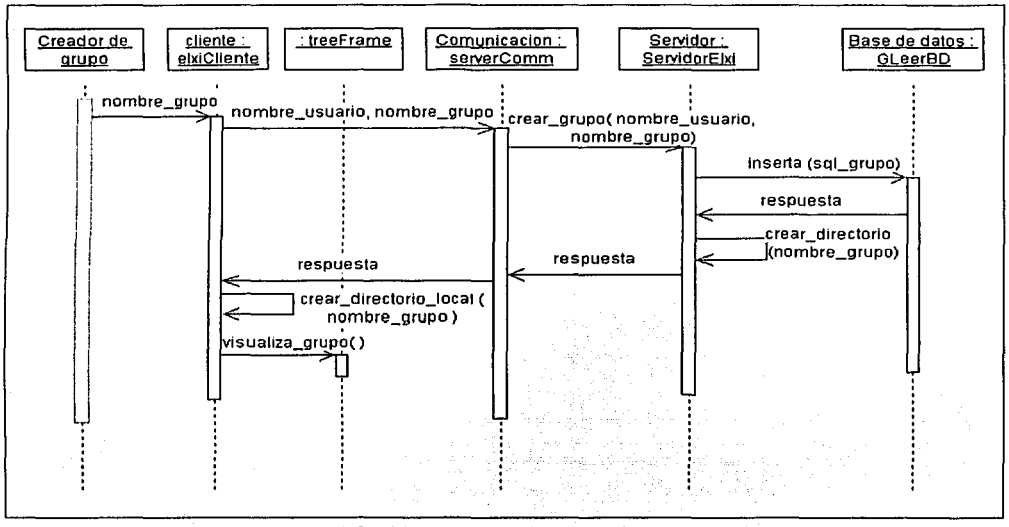


Figura 3.20. Diagrama de secuencia "Crear grupo"

Otra operación que se requiere en un grupo es la de agregar miembros (este caso de uso es mostrado en la figura 3.21). Debido a que el creador es el interesado en formar un grupo de trabajo, tendrá la responsabilidad de especificar la lista de miembros, así como controlar el crecimiento de los miembros del grupo. Esta nueva actividad que puede realizar el colaborador no se contempló en la parte del análisis, por lo que en esta etapa de desarrollo se propone la creación de un nuevo rol nombrado *creador de grupo*.

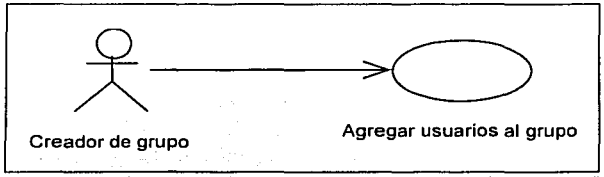


Figura 3.21. Diagrama de caso de uso "Agregar usuarios al grupo".

El diagrama de secuencia para agregar miembros a un grupo se muestra en la figura 3.22. Cuando el servidor atiende la petición de *agregar_miembros*, debe verificar que el colaborador que lo solicita sea el creador del grupo, si es el caso, entonces registra el nombre de los miembros en la base de datos y posteriormente envía una respuesta afirmativa al proceso cliente. A su vez, el servidor se encargará de notificar a los procesos clientes activos, cuyos colaboradores son miembros del grupo, de la operación efectuada.
En el caso de que el colaborador no sea el creador del grupo entonces el cliente recibirá como respuesta un mensaje de operación no válida.

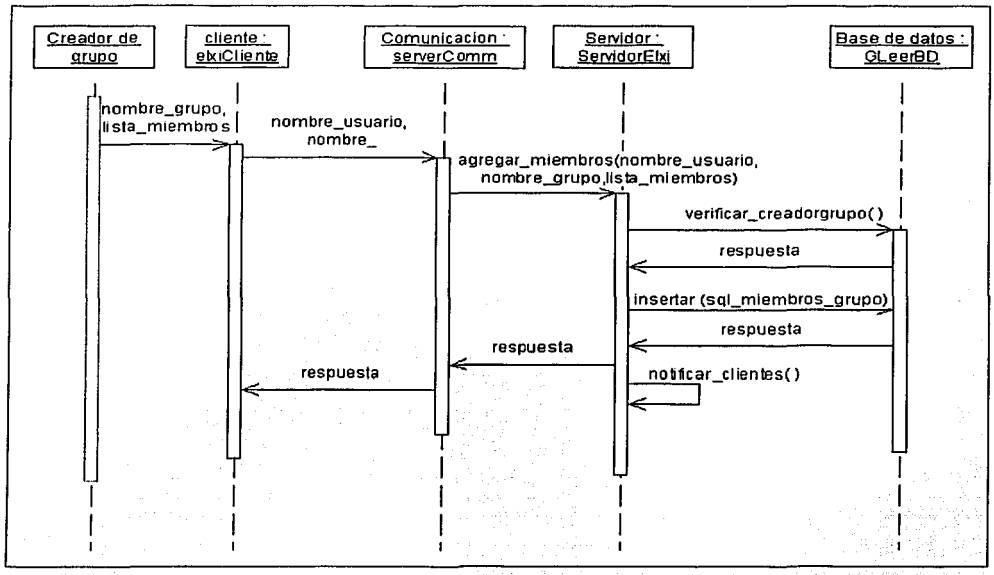


Figura 3.22. Diagrama de secuencia "Agregar usuarios a un grupo".

Debido a que en la sección 3.2.2, se propuso que un documento fuera creado en un grupo y dividido en fragmentos, se requerirá almacenar información como el nombre del documento, el número de fragmentos, el nombre del grupo, así como el nombre del creador. El creador del documento tendrá por omisión el rol de administrador y esta información junto con la anterior será envida por el cliente a través del objeto de tipo *fileObject*. El diagrama de caso de uso de la actividad *crear documento* se muestra en la figura 3.23.

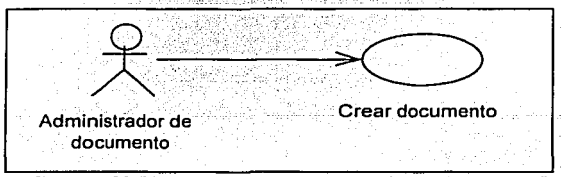


Figura 3.23 Diagrama de caso de uso "Crear documento".

Una vez que el servidor atiende la petición *crear_documento*, éste debe verificar si el colaborador es miembro del grupo en donde se desea crear el documento. Si la respuesta es afirmativa, el nombre del documento, así como los nombres de los fragmentos son registrados en la base de datos, la cual devuelve un número de identificación único para el documento y para cada fragmento. Estos datos son utilizados para crear físicamente el documento a través de la clase *EditDocument*. A continuación, el servidor envía al cliente el identificador del documento, para que lo almacene en el archivo que el objeto de tipo *xmlEdit* va a crear en el sistema de archivos local. Una vez completada esta operación el documento es visualizado en el escritorio de trabajo. El diagrama de secuencia para crear un documento se muestra en la figura 3.24

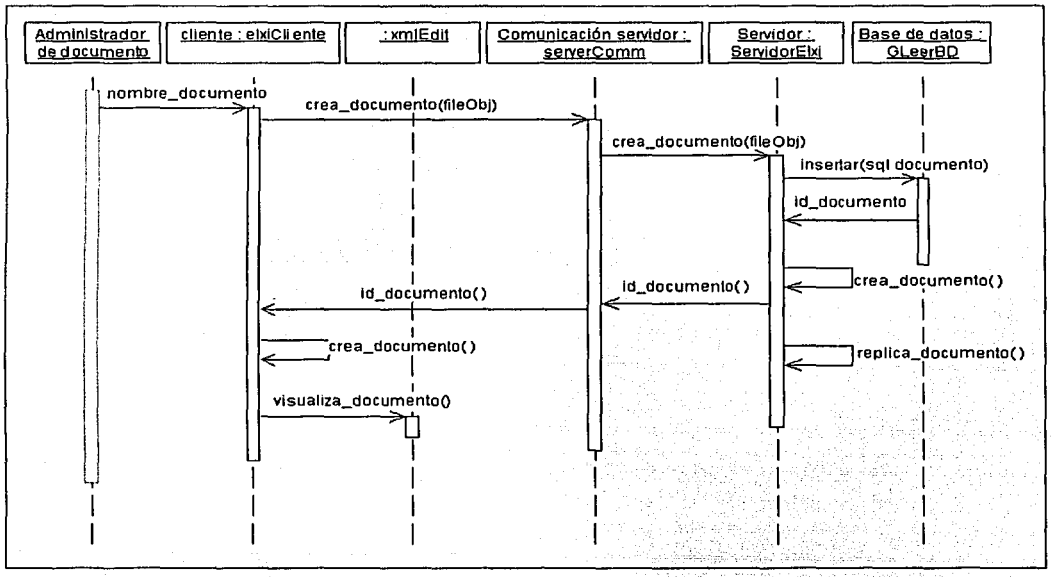


Figura 3.24. Diagrama de secuencia "Crear documento".

Para controlar la obtención de un documento se verá el caso de uso *asignar roles de documento* (ver figura 3.25). El colaborador puede realizar la acción de asignar roles si tiene el rol de administrador del documento. Para que exista un administrador a partir de que el documento se crea, se ha propuesto que por omisión el creador del documento tenga el rol de administrador.

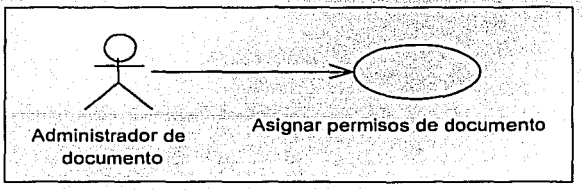


Figura 3.25. Diagrama de caso de uso "Asignar permisos de documento".

En este caso de uso, el colaborador tiene que seleccionar el documento y proporcionar la lista de los miembros del grupo con sus respectivos roles de documento (administrador o autor). En el rol de autor de documento se debe definir explícitamente un permiso, ya que sólo puede adquirir el de obtener el documento, agregar fragmentos al documento, o sin permisos. Posteriormente, el proceso cliente envía la solicitud *permisos_documento* al servidor, el cual verificará que el colaborador tenga el rol de administrador y que los miembros pertenezcan al grupo. Esto se puede observar en el diagrama de secuencia de la figura 3.26. Si el colaborador es el administrador del documento, el servidor se encargará de registrar los roles y permisos de los colaboradores en la base de datos. Enseguida, el proceso servidor mandará al cliente una respuesta de operación completada. Asimismo, notificará a los procesos clientes activos cuyos colaboradores tiene el permiso de obtener el documento, de la operación efectuada. En caso de que el colaborador no sea administrador del documento, no se registrará nada en la base de datos y el cliente recibirá un mensaje de operación no válida.

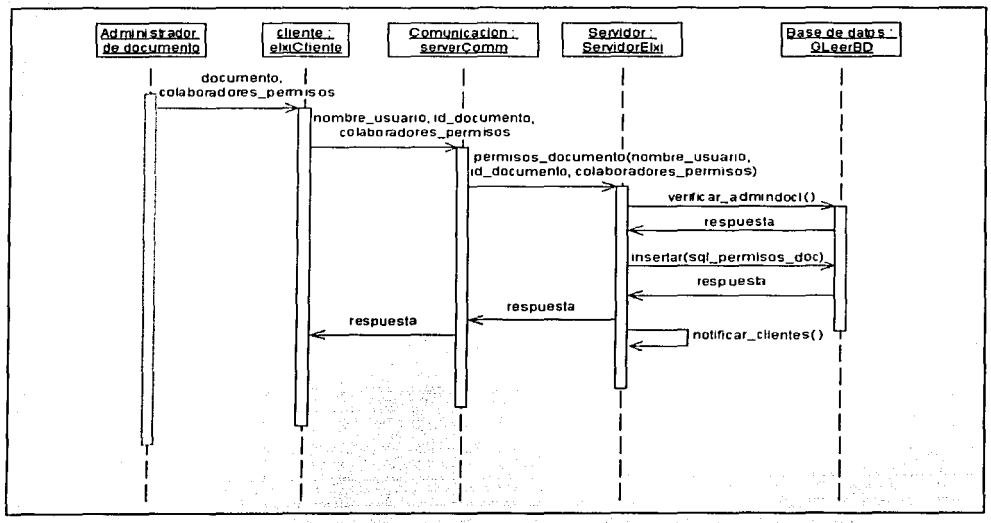


Figura 3.26. Diagrama de secuencia "Asignación de permisos de documento".

Para escribir en un documento, o mejor dicho en un fragmento se debe especificar que autor del documento puede hacerlo. Para ello, es necesario la asignación de roles de fragmento a los autores del documento. Este caso de uso se puede observar en la figura 3.27.

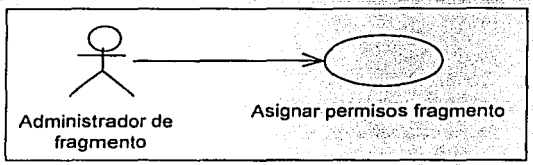


Figura 3.27. Diagrama de caso de uso "Asignar permisos de fragmento".

Los roles sobre fragmentos pueden ser tres, el de administrador, escritor o lector. Un colaborador puede adquirir el rol de administrador de dos maneras. La primera es que otro colaborador con el rol de administrador de fragmento le asigne ese rol. La segunda es que los fragmentos que se crearon junto con el documento sean administrados por el colaborador que creo el documento. Entonces siempre existe un administrador de fragmento.

Para que el administrador de fragmento le asigne roles de fragmento a los autores del documento, debe indicar el fragmento del documento y los autores con sus respectivos roles y permisos. Posteriormente, el proceso cliente envía la solicitud *permisos_fragmento* al servidor, el cual se encargará de verificar que el colaborador sea el administrador del fragmento y que los autores tengan acceso al documento. Si este es el caso, el servidor registrará en la base de datos los permisos de los autores sobre el fragmento. Enseguida, el proceso servidor se encargará de enviar al cliente una respuesta de éxito (ver figura 3.28). En caso de que algún autor no tenga el permiso de obtener el documento, no se le asigna el permiso de fragmento en la base de datos y el proceso cliente recibirá un mensaje de excepción. Finalmente, el servidor notificará a los procesos clientes activos, cuyos colaboradores son los que se les otorgaron permisos sobre el fragmento, de la operación efectuada.

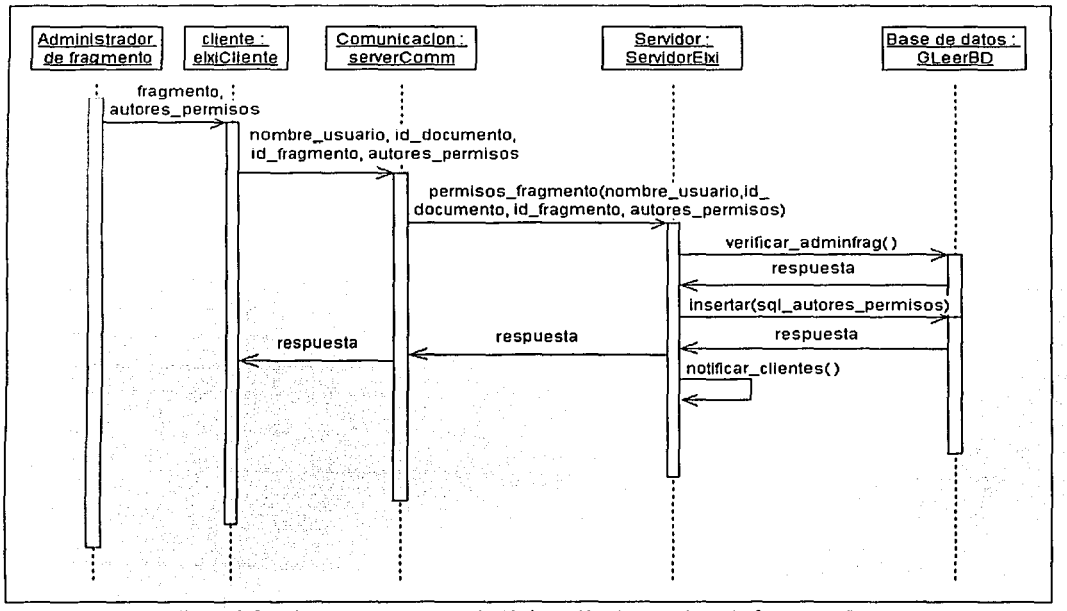


Figura 3.28.Diagrama de secuencia "Asignación de permisos de fragmento".

Otro caso de uso que surge en el diseño es el bloqueo de un fragmento (ver figura 3.29). Un autor puede escribir o modificar el contenido de en un fragmento siempre que tenga el rol de escritor y además se le haya concedido el bloqueo. El autor adquiere el rol de escritura cuando el

administrador del fragmento se lo otorga. Para obtener el bloqueo necesita seleccionar esta opción dentro del editor de ELXI indicando además el tiempo que lo desea bloquear.

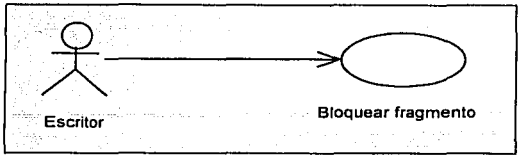


Figura 3.29. Diagrama de caso de uso "Bloquear fragmento".

Una vez que el colaborador establece el tiempo de bloqueo, el cliente envía al servidor la solicitud *bloquear_fragmento* junto con el nombre del colaborador, el número de identificación del documento y del fragmento (ver figura 3.30). Posteriormente, el servidor verificará que el colaborador tenga el rol de escritor y que el fragmento no este bloqueado por otro escritor. Si el fragmento no esta boqueado, el servidor registrará el bloqueo en la base de datos y enseguida le mandará al cliente una respuesta de éxito. Después, el cliente a través del objeto del tipo *xmlEdit* habilitará el fragmento para la edición. Por otra parte, el proceso servidor notificará a los procesos clientes activos, cuyos colaboradores son autores del documento, de la operación efectuada.

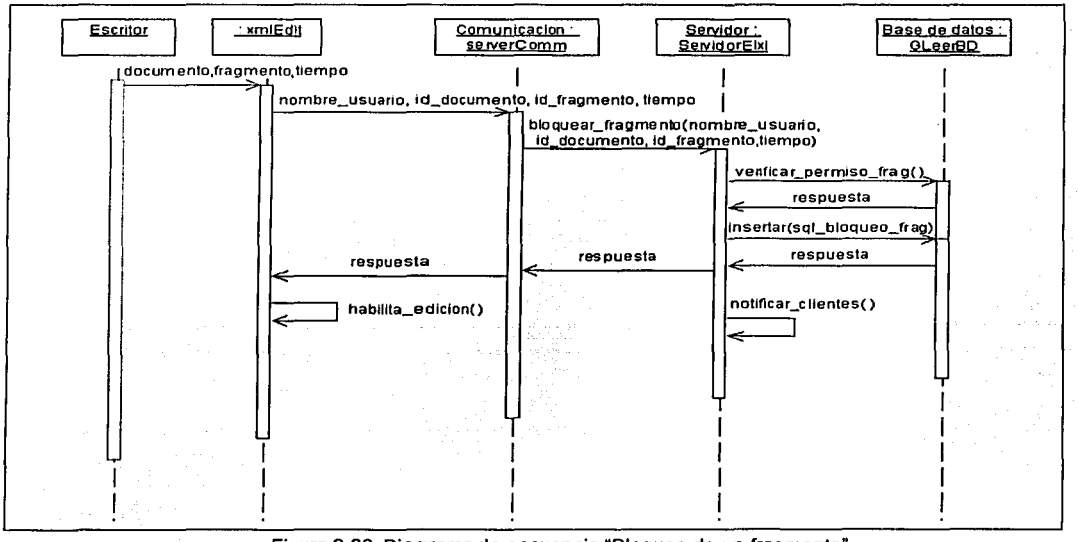


Figura 3.30. Diagrama de secuencia "Bloqueo de un fragmento"

En caso de que el fragmento haya sido bloqueado por otro autor, se tienen dos posibles soluciones. La primera es que el cliente muestre un mensaje anunciando que el bloqueo lo tiene otro autor y por esa razón no se le ha concedido. Es probable que el autor lo intente más tarde pero también se puede dar la situación de que otro autor se anticipe y obtenga el bloqueo. Por lo tanto, el autor que lo intento antes tendrá que esperar otro periodo de tiempo más.

La segunda solución es mostrar el mensaje anteriormente mencionado y además brindar la opción de reservar un lugar en una lista de espera para obtener el bloqueo, de tal manera que cuando se libere, el primer autor que hizo la reservación, es decir, el primero de la lista de espera obtenga el bloqueo.
De las dos anteriores soluciones se optó por la segunda, debido a que de esta forma no se le niega el servicio al colaborador, además de que se le ofrece la oportunidad de obtener el bloqueo sin necesidad de volver a ejecutar la misma operación. Ver caso de uso de la figura 3.31.

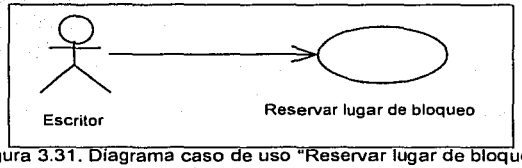


Figura 3.31. Diagrama caso de uso "Reservar lugar de bloqueo.

El diagrama de secuencia para la reservación del bloqueo se muestra en la figura 3.32. El colaborador debe indicar el documento, el fragmento y el tiempo de bloqueo. Una vez realizado, el cliente le enviará al servidor la solicitud *reservar_bloqueo*. Después el proceso servidor se encargará de verificar que el colaborador tenga el rol de escritor en el fragmento. Si es el caso, el proceso servidor ingresará en una lista de espera la interfaz del objeto cliente, junto con la información de bloqueo. Posteriormente, le enviara de regreso al cliente una respuesta con la posición en la que se encuentra el colaborador en la lista de espera para obtener el bloqueo del fragmento.
El servidor debe atender las peticiones de acuerdo al orden en que van llegando, por lo que la lista de espera será implementada mediante una estructura de tipo FIFO (First-in First-out), de tal manera que la primera solicitud en llegar sea la primera en ser atendida.

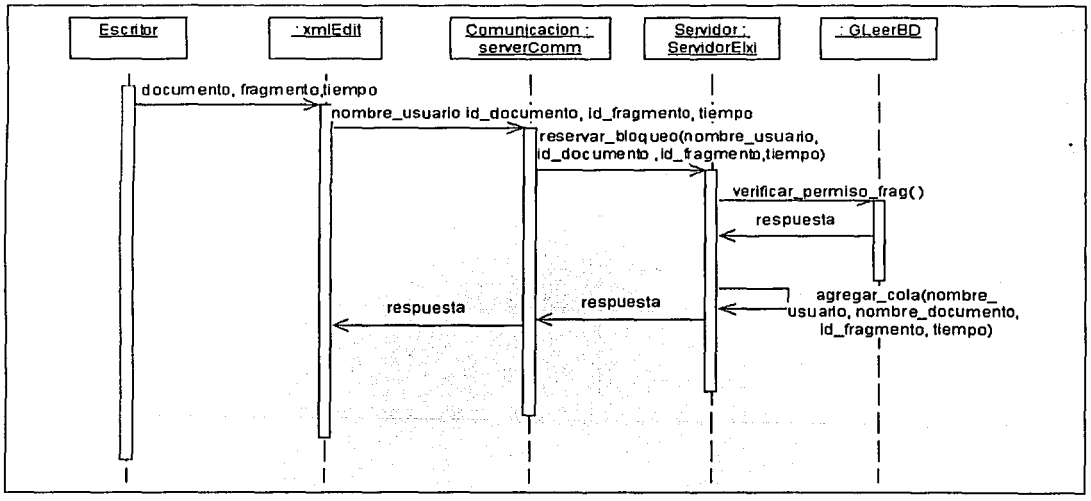


Figura 3.32. Diagrama de secuencia "Reservación de lugar para bloqueo de fragmento".

Es posible que el colaborador haya terminado de editar el fragmento antes del tiempo que estableció en el sistema, en este caso tiene la opción de quitar el bloqueo permitiendo que otro escritor pueda obtenerlo. Este caso de uso se muestra en la figura 3.33.

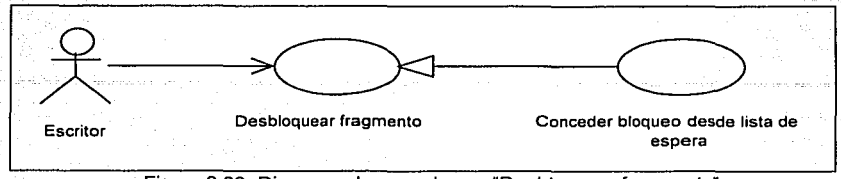


Figura 3.33. Diagrama de caso de uso "Desbloquear fragmento".

El diagrama de secuencia para desbloquear un fragmento se muestra en la figura 3.34. El cliente es el encargado de enviar al servidor el nombre del usuario, el identificador del documento y del fragmento. Posteriormente, el servidor verifica en la base de datos que el colaborador tenga el bloqueo, si es así, entonces quitará el bloqueo sobre ese fragmento (indicado también en la base de datos). Después, devuelve una respuesta al proceso cliente quien se encargará de deshabilitar la edición sobre el fragmento mediante el objeto de tipo *xmlEdit*. Asimismo, el servidor se encargará de revisar la lista de espera con el fin de conceder el bloqueo a aquella solicitud que haya hecho referencia al fragmento que se acaba de liberar. Sino existe alguna solicitud en espera, el servidor notificará a los procesos clientes activos, cuyos colaboradores son autores del documento, de la liberación del bloqueo.

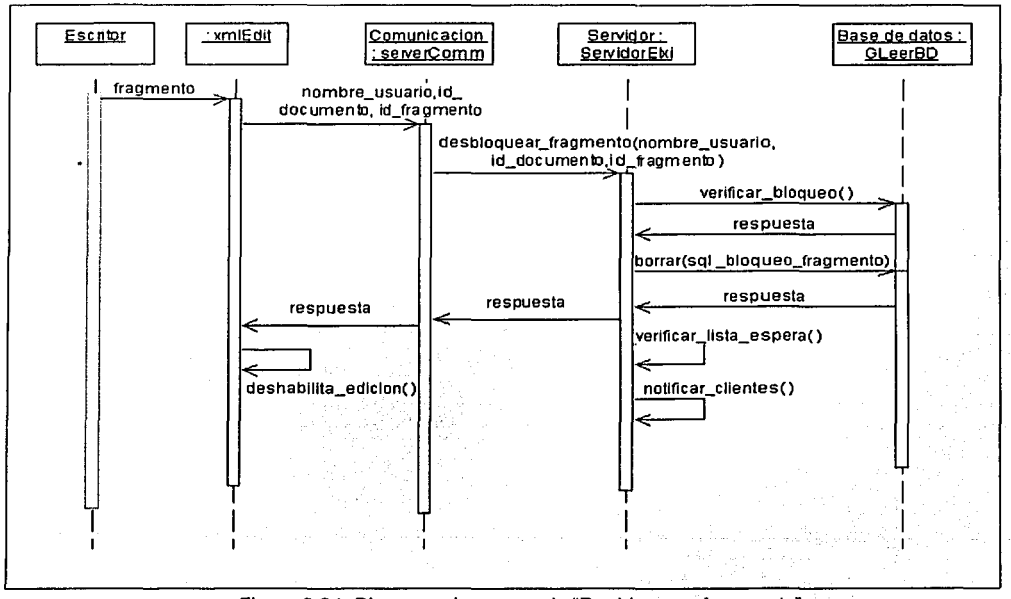


Figura 3.34. Diagrama de secuencia "Desbloquear fragmento".

En el caso de que exista una solicitud en la lista de espera cuando el fragmento sea desbloqueado, el proceso servidor registrará el nuevo bloqueo en la base de datos para el colaborador de esa solicitud. Asimismo le enviará un mensaje a su proceso cliente, especificando que el bloqueo ha sido otorgado. Posteriormente, el cliente mediante el objeto de tipo *xmlEdit* habilitará la edición del fragmento y el colaborador con el rol de escritor podrá empezar a escribir.

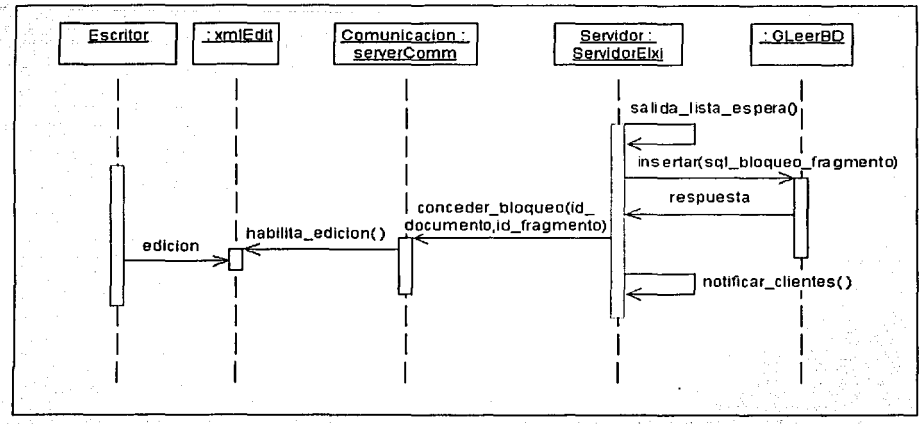


Figura 3.35. Diagrama de secuencia "Conceder bloqueo desde la lista de espera".

Se puede dar el caso de que el tiempo de bloqueo expire (ver diagrama de la figura 3.36). El proceso servidor al detectar esta situación, se encargará de quitar el bloqueo al colaborador correspondiente. Además, tendrá la tarea de verificar si existe una solicitud de bloqueo en la lista de espera y atenderla.

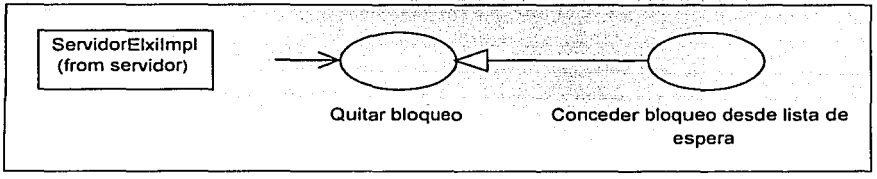


Figura 3.36. Diagrama de caso de uso "Quitar bloqueo".

El diagrama de secuencia para este caso se muestra en la figura 3.37 y consiste en lo siguiente. Cuando ya finalizó el periodo de bloqueo para un determinado fragmento, el servidor borra el bloqueo de la base de datos y envía al proceso cliente el mensaje de tiempo de bloqueo expirado. Posteriormente, el proceso cliente mediante el objeto de tipo *xmlEdit* deshabilitará la edición sobre el fragmento. Así también, el proceso servidor se encargará de notificar a los procesos clientes activos, cuyos colaboradores son autores del documento, de la operación efectuada.

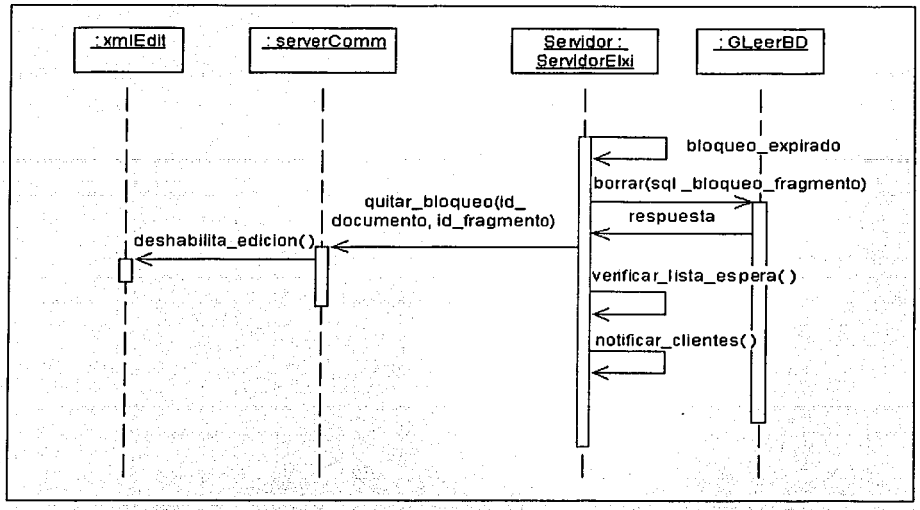


Figura 3.37. Diagrama de secuencia "Quitar Bloqueo".

Debido a que el contenido de un documento es almacenado en fragmentos, corresponde analizar el caso de uso guardar fragmento (ver figura 3.38). Para guardar el contenido de un fragmento, el colaborador, que tiene el rol de escritor, debe tener el bloqueo para poder seleccionar la opción de guardar, la cual es habilitada por el objeto *xmlEdit*.

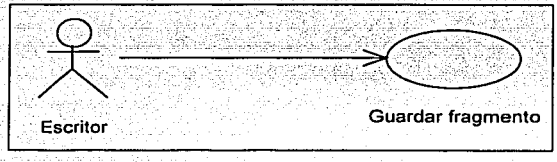


Figura 3.38. Diagrama de casó de uso "Guardar fragmento".

El proceso cliente envía al servidor el fragmento como un objeto. Una vez que el servidor recibe la solicitud *guardar_fragmento*, éste se encargará de verificar que el colaborador tenga el bloqueo sobre el fragmento, si es así, procederá a buscar en el sistema de archivos el documento respectivo. Este documento es analizado sintácticamente por el objeto de tipo *Parser* quien se encargará de verificar que el documento XML sea válido, posteriormente el objeto *EditDocumento* crea la estructura de árbol del documento con sus respectivos objetos fragmentos. Enseguida, localizará el objeto fragmento mediante su número de identificación para sobrescribir su contenido. Después el objeto documento es guardado en el respectivo archivo. Enseguida, el servidor envía una respuesta de operación exitosa al cliente, quien se encargará de guardar el fragmento en el correspondiente archivo local a través del objeto de tipo *xmlEdit*. Así también, el proceso servidor se encargará de replicar el fragmento a los procesos clientes activos, cuyos colaboradores son autores del documento. Ver figura 3.39.

En caso de que el colaborador no tenga el bloqueo o que el fragmento no se haya podido guardar en el documento, todas las operaciones involucradas en el proceso de guardar documento son anuladas y el cliente obtiene como respuesta un mensaje de operación no ejecutada.

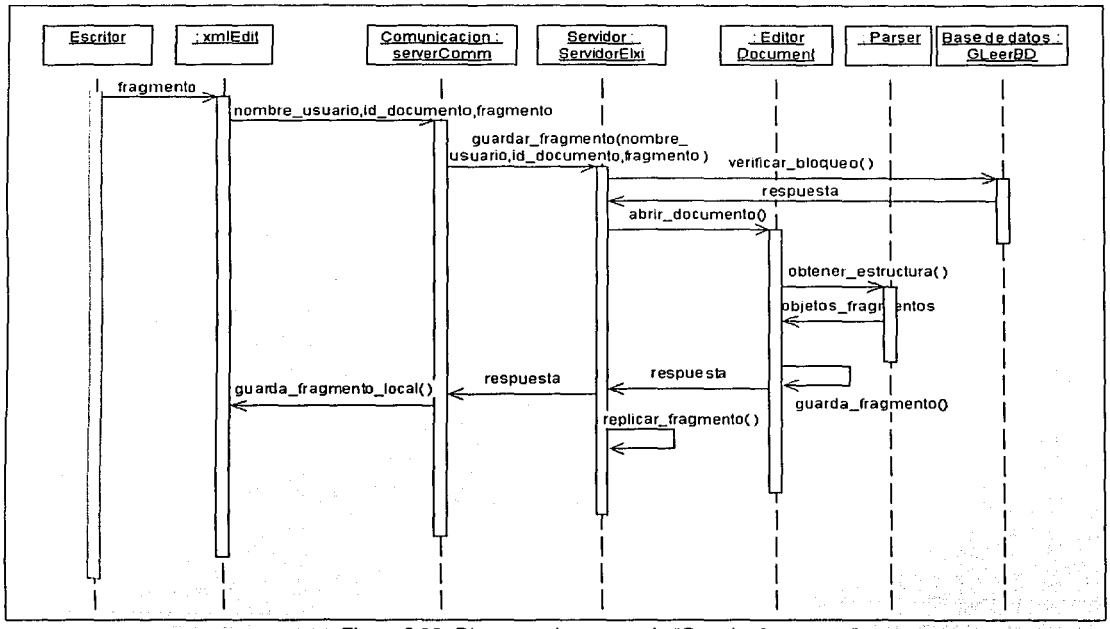


Figura 3.39. Diagrama de secuencia "Guardar fragmento".

Como anteriormente se mencionó, al crear un documento también se especifica el número de fragmentos que lo compondrán, pero no quiere decir que este número sea fijo, ya que la estructura de un documento puede variar durante su elaboración. Para agregar un fragmento a un documento, un colaborador debe tener el rol de autor junto con el permiso de *agregar fragmento*. El caso de uso es mostrado en la figura 3.40.

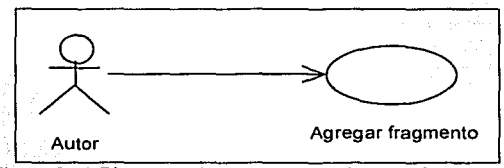


Figura 3.40. Diagrama de caso de uso "Agregar fragmento".

El diagrama de secuencia para agregar un fragmento se muestra en la figura 3.41. Una vez que el colaborador ha seleccionado la opción de agregar fragmento en el objeto de tipo *xmlEdit*, la parte cliente enviará al servidor la petición de *agregar_fragmento*. A su vez, el proceso servidor se encargará de verificar los permisos de documento del colaborador y si tiene el permiso de agregar

fragmento entonces se insertará un nuevo registro en la base de datos referente al fragmento. Al efectuar esta operación, la base de datos le regresará al servidor un número de identificación de fragmento único. Posteriormente, el servidor localizará el documento dentro del sistema de archivos, el cual será validado mediante el objeto de tipo *Parser*. Enseguida, el objeto de tipo *EditorDocument* creará la estructura de árbol del documento, así como también, un objeto de tipo fragmento. Este objeto será insertado en la estructura de árbol documento junto con su número de identificación y guardado en el archivo correspondiente. Si estas operaciones se ejecutaron con éxito, el servidor enviará a la parte cliente el número de identificación del fragmento. Una vez que el cliente lo obtiene, mediante el objeto de tipo *xmlEdit* creará un objeto fragmento, el cual contendrá el número de identificación, que será guardado en el archivo local correspondiente. Finalmente, el documento será visualizado en el espacio de trabajo. Por otra parte, el proceso servidor se encargará de replicar el fragmento a los procesos clientes activos, cuyos colaboradores son autores del documento.

Si el colaborador no tiene el permiso de agregar un fragmento al documento, o bien, si la operación de insertar el registro del fragmento en la base de datos o el de guardar el objeto fragmento en el documento fracasan, entonces todas las operaciones que se realizaron durante el proceso de agregar fragmento son anuladas y el cliente obtiene una respuesta de operación no ejecutada.

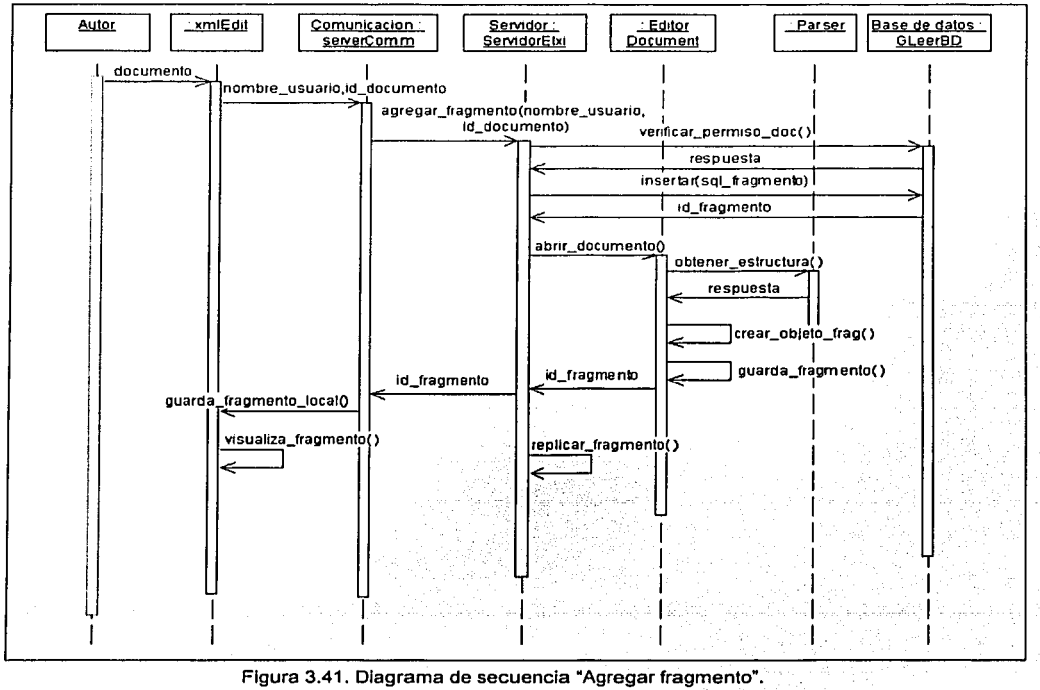


Figura 3.41. Diagrama de secuencia "Agregar fragmento".

Las operaciones de borrar fragmento, borrar documento, borrar usuario del grupo y borrar grupo no pueden ser realizadas por cualquier colaborador. Los roles de creador de grupo, administrador de documento y administrador de fragmento, han sido creados con el fin de que algunos colaboradores con estos privilegios controlen estas acciones. Además, la operación de borrar no es sólo desaparecer una entidad, también hay que analizar cuales serán las consecuencias.

Para borrar un fragmento, sólo lo podrá hacer el administrador del fragmento siempre que no exista un bloqueo en él. Para establecer este enunciado consideramos el siguiente ejemplo. Existe un autor que se encuentra editando un fragmento y a su vez el administrador de ese fragmento decide borrarlo. Entonces el fragmento desaparece y la aportación del autor es interrumpida. Para este caso, se observó que se debería dar la oportunidad de que el autor concretará su idea. Además, también debería ser enterado del borrado del fragmento, lo que le daría la oportunidad de rescatar sus aportaciones. El diagrama de secuencia para borrar un fragmento involucra las mismas entidades del diagrama de secuencia "Agregar fragmento" de la figura 3.41. Las operaciones que varían son con respecto a la base de datos, ya que para este caso se realizaría el borrado del registro del fragmento en la base de datos y también se requeriría removerlo físicamente del archivo.

En el caso de borrar el documento se considera el siguiente panorama. Es posible que existan varios administradores de fragmento (ninguno de ellos es el administrador del documento) interesados en la elaboración del documento, Sin embargo, el administrador del documento que no ha participado en la edición decide borrar el documento, entonces los demás administradores de fragmento verán afectado su trabajo. Por ello, se ha establecido que un documento pueda ser borrado, si el único autor asociado a él es el administrador del documento y también es el administrador de los fragmentos existentes. El diagrama de secuencia para borrar un documento involucra las mismas entidades del diagrama de secuencia "Crear documento" de la figura 3.23. Las operaciones que varían son con respecto a la base de datos, ya que para este caso se realiza el borrado del registro del documento junto con sus fragmentos en la base de datos y también se requiere removerlo físicamente del sistema de archivos.

Por otra parte, un grupo puede ser borrado si el único miembro que existe en el grupo es el que lo creo y además no existe un documento creado. No es conveniente que sean borradas las aportaciones de aquellos colaboradores que han creado documentos en un grupo. De igual manera, se sugiere que los miembros no sean removidos por el creador del grupo si están asociados a un documento. El diagrama de secuencia para borrar un grupo involucra las mismas entidades del diagrama de secuencia "Crear grupo" de la figura 3.20. Las operaciones que varían son con respecto a la base de datos, ya que para este caso se realiza el borrado del registro del grupo en la base de datos y también se requiere removerlo físicamente del sistema de archivos de la parte servidor y cliente. Para borrar un miembro de un grupo, tan solo hay que borrar de la base de datos el registro que relaciona el colaborador con el grupo.

En la figura 3.42 se muestran todos los casos de uso que se trataron en la etapa de análisis para comenzar a desarrollar el mecanismo de control de concurrencia.

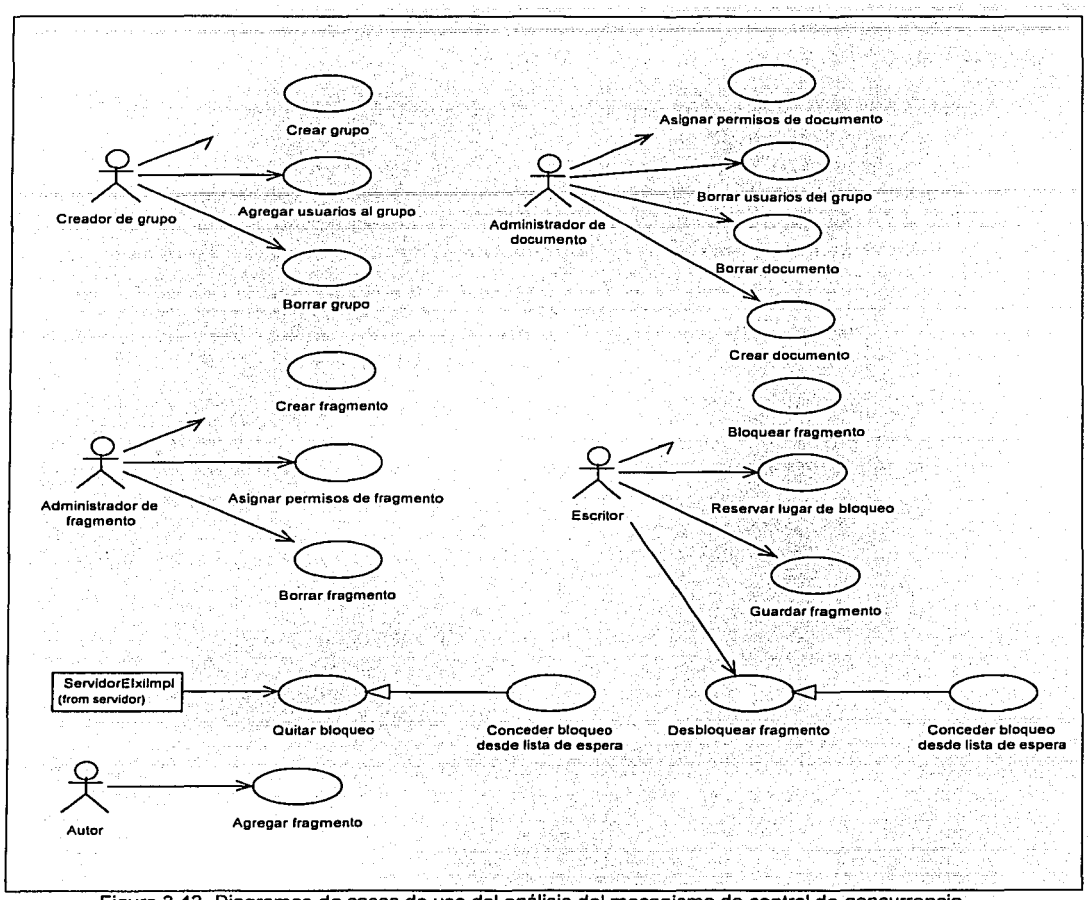


Figura 3.42. Diagramas de casos de uso del análisis del mecanismo de control de concurrencia.

De esta manera se ha concluido la parte del diseño del mecanismo de control de concurrencia, en la siguiente sección se realizará la implementación del mismo, reestructurando algunas entidades involucradas.

3.22 Implementación

Debido a que el control de concurrencia que se propone esta basado en una nueva estructura de documento, asignación de responsabilidades mediante roles de documento y fragmento, y la implementación del mecanismo de bloqueo en la edición concurrente, algunas entidades del sistema ELXI sufrirán algunos cambios.

La base de datos, además de almacenar la información de los colaboradores y de los documentos, tendrá que organizar también la de los grupos, roles y fragmentos. Por lo tanto, se propone en la figura 3.43 la nueva estructura de la base de datos ELXIDB.

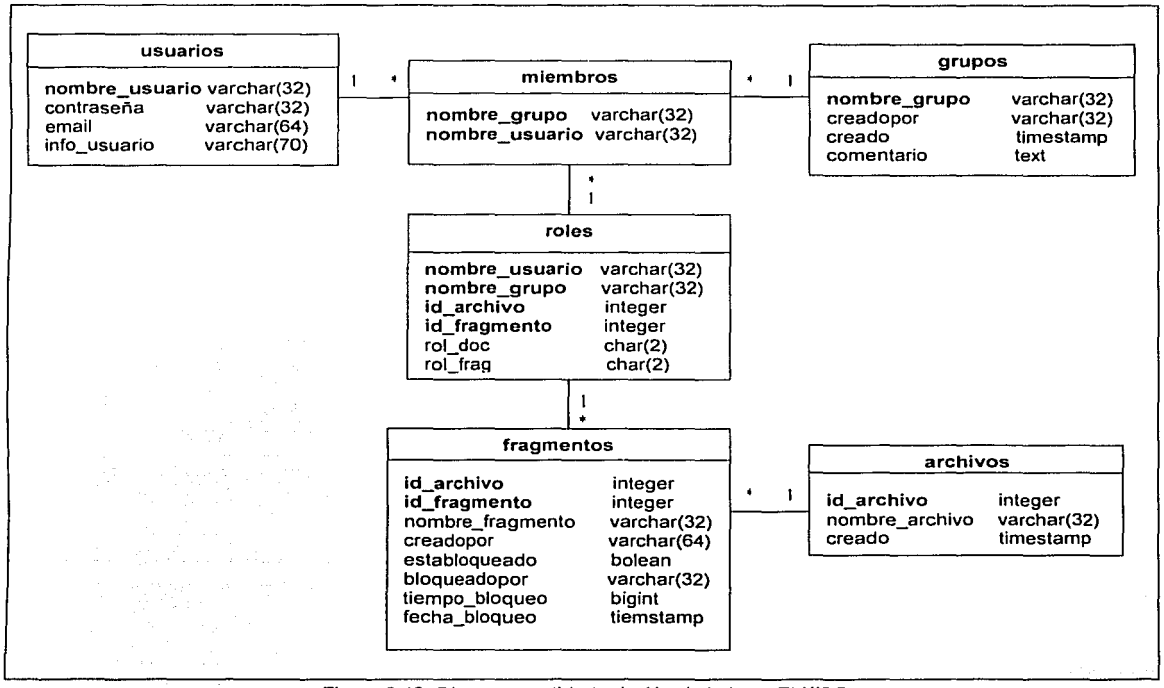


Figura 3.43. Diagrama entidad relación de la base ELXIDB.

La tabla *miembros* rompe la relación muchos a muchos entre la entidad *usuarios* y *grupos*, estableciendo qué colaborador pertenece a qué grupo. La tabla *archivos* almacena el nombre del documento, así como su fecha de creación. Esta tabla tiene una relación de uno a muchos con la tabla *fragmentos*, la cual además de identificar a cada fragmento como único, establece cuatro campos utilizados para el bloqueo. El campo *establoqueado* indica si ha sido bloqueado el fragmento mediante los valores de verdadero o falso. El campo *bloqueadopor* almacena el nombre del usuario que lo bloqueo, esta información fomenta la conciencia en grupo. El campo *tiempo_bloqueo* almacena la duración del bloqueo en segundos. Y por último la *fecha_bloqueo* indica desde cuando fue bloqueado el fragmento.

La relación entre miembros, archivos y fragmentos la establece la *tabla roles*, en la cual, mediante el campo *rol_doc* se establece el permiso que tiene un usuario sobre un documento. Los posibles valores de este campo son '*AD*' (Administrador de Documento), '*LD*' (Obtener Documento) o '*ED*' (Agregar Fragmento). Así también, especifica el rol de fragmento en el campo *rol_frag* , cuyos valores pueden ser '*AF*' (Administrador de Fragmento), '*EF*' (Escritor de Fragmento) o '*LF*' (Lector de Fragmento). De los roles mencionados faltaría el de creador de grupo, sin embargo este no será identificado en la misma tabla, sino en el campo *creadopor* de la tabla *grupos*, el cual almacenará el nombre del colaborador.
Asimismo, se realizó una modificación en la estructura de archivos. Cuando sea creado un grupo, también será creado un directorio con el nombre del grupo en el sistema de archivos. Este directorio *grupo* se encontrará localizado dentro del directorio *datos* donde reside la aplicación ELXI del proceso servidor. Debido a que un documento puede ser creado únicamente por un colaborador que es miembro de un grupo, entonces el documento también forma parte de un grupo. Por lo tanto, el directorio *grupo* contendrá los documentos XML que se creen en ese grupo. En la figura 3.44, se observan dos directorios *grupo,* Sistemas Distribuido y CSCW, los cuales almacenan archivos con extensión xml.

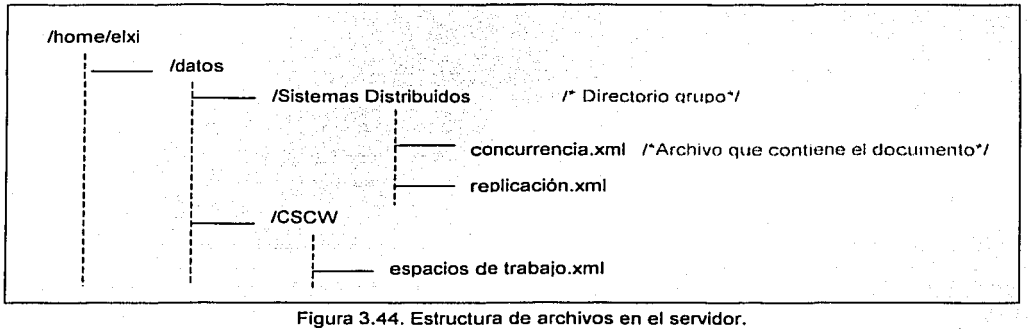


Figura 3.44. Estructura de archivos en el servidor.

En la parte cliente se maneja una estructura de archivos similar. La diferencia radica en que se crea un directorio para cada colaborador. Este directorio contendrá el archivo de información y los directorios *grupo* a los que pertenece. A su vez, los directorios *grupo* almacenarán los archivos a los que puede acceder el colaborador. En la figura 3.45, se observa que el directorio *silvia* es el directorio del colaborador donde residen los grupos y archivos que puede acceder.

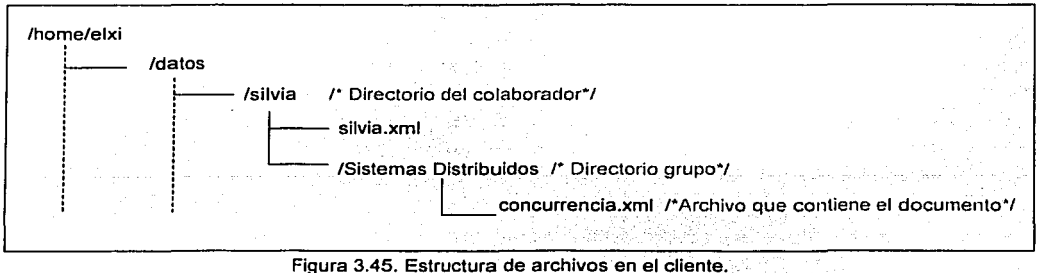


Figura 3.45. Estructura de archivos en el cliente.

Igualmente, la estructura de los documentos tuvo un cambio. Como se puede observar en la figura 3.46.a, la DTD para un archivo de tipo documento esta constituida por el elemento *documento* que a su vez contiene un elemento *fragmento*. El archivo de tipo documento se muestra en la figura 3.46.b. Las palabras que sobresalen con negro son las etiquetas XML que identifican cada parte del archivo y son del tipo *documento* y *fragmento*.

```
<!ELEMENT documento (fragmento+)>
<!ATTLIST documento
    id_file CDATA #REQUIRED>

<!ELEMENT fragmento (#PCDATA)>
<!ATTLIST fragmento
    titulo CDATA #REQUIRED
    id_fragment CDATA #REQUIRED
    tipo CDATA "">
```

(a)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE documento SYSTEM "documento.dtd">
<documento id_file="2">
<fragmento id_fragment="1" tipo="title" titulo="Ttitulo">CONCURRENCIA EN LA ELABORACION DE DOCUMENTOS </fragmento>
<fragmento id_fragment="2" tipo="authors" titulo="Autores">Dr. Manuel Romero, Ing. Silvia Ramirez Garcia</fragmento>
<fragmento id_fragment="3" tipo="abstract" titulo="Resumen">Actualmente, existe una mutidisciplina que investiga el campo del potencial de la computadora para ayudar en el trabajo en grupo conocida como el trabajo colaborativo asistido por computadora (CSCW)... </fragmento>
<fragmento id_fragment="4" tipo="section" titulo="Introducción">El propósito de esta tesis es incorporar a la arquitectura del sistema de edición colaborativa ELXI (Editor coLaborativo de documentos XML en Internet), un mecanismo de control deconcurrencia que permita que diferentes participantes (autores y revisores) puedan acceder un mismo documento de manera remota para editarlo o revisarlo. </fragmento>
<fragmento id_fragment="5" tipo="section" titulo="Contribución">Actualmente, en el Depto. de Ciencias de la Computación del IIMAS, UNAM, bajo el proyecto de investigación &amp;quot;Ambiente Multipropósito para la Edición Colaborativa en Intranet e Internet&amp;quot; apoyado por el CONACYT se está desarrollando un nuevo editor llamado ELXI (Editor coLaborativo de documentos XML en Internet). Uno de sus notables aportaciones es el almacenamiento de la información en archivos. Estos son documentos creados en forma estructurada mediante el lenguaje de marcado extensible (XML), permitiendo que su manejo sea como el de un objeto donde el acceso y almacenamiento de datos se realiza de forma eficiente. </fragmento></documento>
```

(b)

Figura 3.46. (a) DTD de un documento. (b) Archivo de tipo documento que contiene un identificador de archivo y contenido.

El archivo que contiene la información del usuario también ha sido reestructurado. Su DTD contiene elementos que hacen referencia a las entidades grupo, documento y fragmento (ver figura 3.47.a). El archivo xml contiene etiquetas en negro que hacen referencia a la información de acuerdo a la entidad (ver figura 3.47.b). La etiqueta *group name* indica el nombre del grupo al que puede acceder el usuario y la etiqueta *file* el nombre del documento. La etiqueta *frag* hace referencia a datos del fragmento del documento como el número de identificación (*id_fragment*), el nombre del fragmento (*name frag*), el usuario que los creó (*createBy*) y si está bloqueado (*locked*). Hay que recordar que este tipo de archivo es utilizado cuando el proceso cliente no realiza una conexión con el servidor, es decir el colaborador está trabajando de manera local.

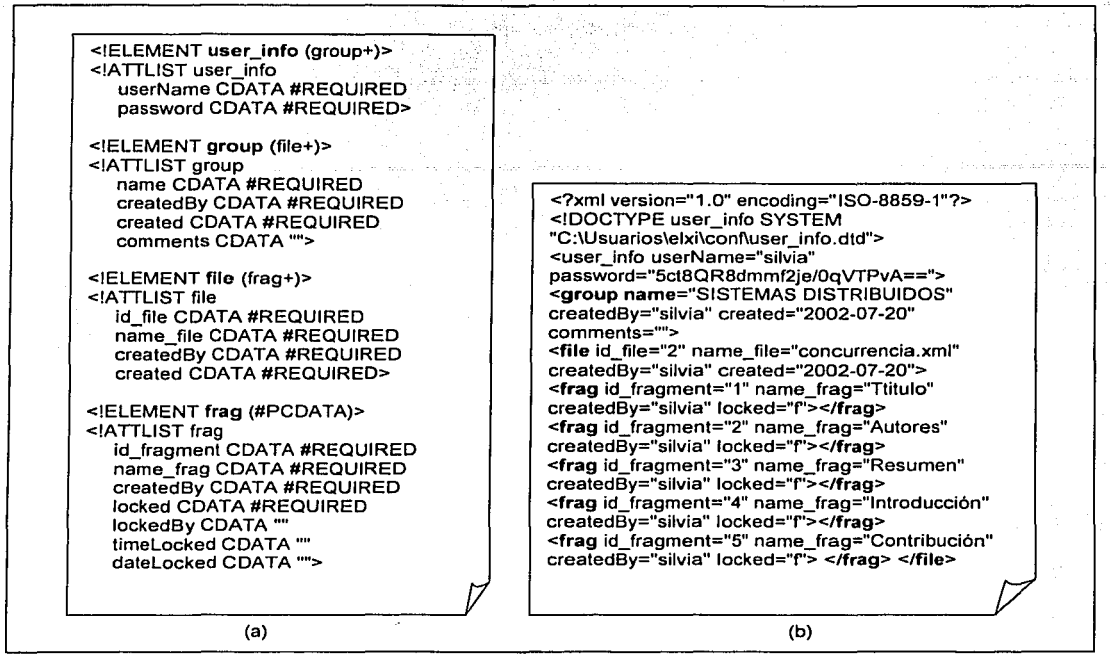


Figura 3.47. (a) DTD de un archivo de tipo user_info. (b) Estructura y contenido de un archivo de tipo user_info .

Ahora corresponde mostrar cuáles son las nuevas aportaciones y modificaciones que se realizaron en la programación de ELXI. En la figura 3.48 se pueden observar las nuevas clases de ELXI utilizadas para implementar el mecanismo de control de concurrencia.

Con respecto a la entidad grupo, se diseñó la clase *systemGroups* para crearla o borrarla. Y para agregar o remover un miembro de un grupo se utiliza la clase *groupUsers*. La clase *DocumentUser* define los métodos necesarios para establecer los roles de documento y la clase *systemPermissions* para los roles de fragmento.

Para bloquear un fragmento se diseñó la clase *timeDialog*, la cual crea un objeto ventana que solicita el tiempo de bloqueo para el fragmento seleccionado. Para almacenar el contenido del fragmento se creó la clase *fragObj*, la cual define entre sus atributos, el identificador del fragmento y el documento al que pertenece, nombre del fragmento, nombre del creador, estado (bloqueado), tiempo de bloqueo, fecha de bloqueo y quien lo bloqueo. Esta clase fue agregada al paquete *util*. La clase *fileObj* fue modificada, ya que en ella ya no se especifica el atributo que almacena el texto. Ahora, sólo hace referencia a los fragmentos que contiene el documento.

Cuando un usuario decide liberar el bloqueo o cuando éste expira, el contenido del fragmento no se guarda automáticamente, por lo que se incorporó una opción en donde se le avisará al usuario que le quedan 10 min. para perder el bloqueo. Para implementarla esta opción, se integró otra clase llamada *waitThread*, con el objetivo de crear hilos que registren el tiempo transcurrido del

bloqueo en la parte cliente. De esta manera el proceso cliente le informa al usuario final que le restan 10 min. para guardar sus aportaciones.

La clase *lockedThread* también es utilizada para crear hilos que lleven el transcurso del tiempo de bloqueo. Sin embargo, estos son utilizados por el proceso servidor para remover el bloque de la base de datos y notificar a los clientes activos de que el bloqueo de un fragmento ha terminado.

Para crear la lista de espera donde se almacenan las solicitudes de bloqueo se incorporó el paquete *Queue*, el cual contiene las clases *LinkedQueue*, *LinkedQueueElement* y *LinkedIterator* que permiten que la lista de espera sea una estructura de tipo FIFO. La clase *ObjectQueue* define un objeto de la lista.

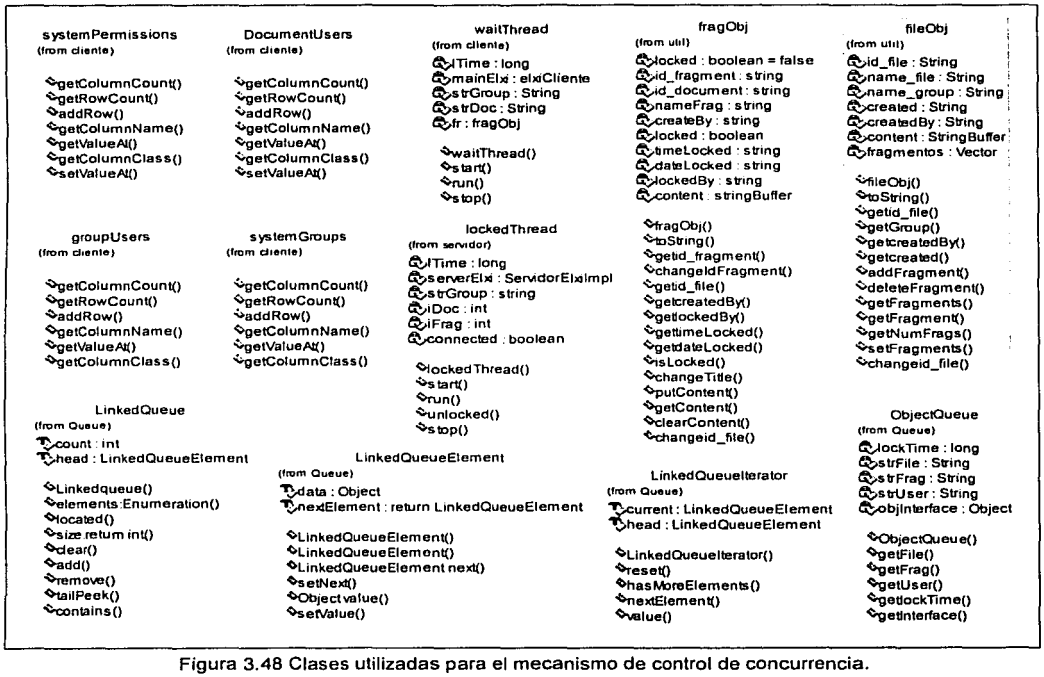


Figura 3.48 Clases utilizadas para el mecanismo de control de concurrencia.

En la figura 3.49 se muestra el diagrama de clases de ELXI, en la cual se incorporan la nuevas clases y sus relaciones con las demás. Se puede observar que las clases *systemGroups*, *groupUsers*, *DocumentUser* y *systemPermissions*, a partir de las cuales crean objetos ventana, establecen una relación con la clase *elxiCliente* Esto es debido a que los objetos ventana son ubicados dentro del espacio de trabajo que define *elxiCliente*. En cambio, las clases *fileObj*, *fragObj* y *timeDialog* tienen una relación directa con la clase *xmlEdit*, ya que están muy involucrados con el manejo de objetos de tipo documento y fragmento, así como del bloqueo. Asimismo la clase *fileObj* y *fragObj* guardan una relación con la clase *ServidorElxiImpl*, la cual también le concierne el acceso y almacenamiento de documentos. De igual manera, la clase *lockedThread* y la interfaz *LinkedQueue* que llama a los métodos de las clases

LinkedQueueElement, *LinkedIterator* y *ObjectQueue* , mantienen una relación con la clase *ServidorElxiImpl*.

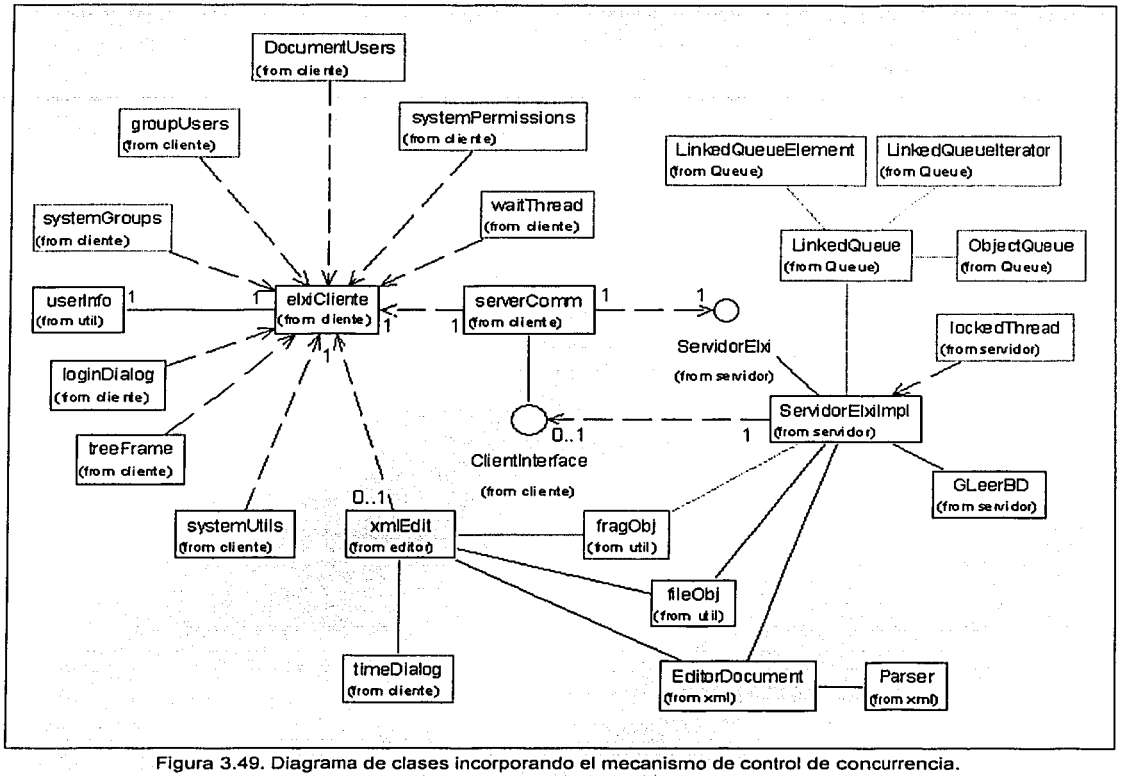


Figura 3.49. Diagrama de clases incorporando el mecanismo de control de concurrencia.

A continuación se explicará parte del código implementado, así como algunas estrategias de programación empleadas.

Como anteriormente se mencionó, un hilo es una secuencia de llamadas que se ejecutan independientemente de otras, y es posible que al mismo tiempo compartan recursos del sistema tales como archivos, memoria, así como objetos construidos dentro de un programa.

Un hilo en JAVA es un objeto que se construye a partir de la clase *java.lang.Thread* y cuando se crea tiene una memoria de trabajo (una abstracción de los caches y de los registro), en la cual se almacenan valores. Algunas cuestiones importantes que surgen bajo este ambiente concurrente son las de determinar que instrucciones deben tener efectos indivisibles, bajo qué condiciones son visibles los efectos de un hilo sobre otro y cuando los efectos de las operaciones sobre un hilo pueden aparecer fuera de orden.

En el lenguaje de programación JAVA, existe un mecanismo de bloqueo que puede resolver problemas de atomicidad, visibilidad y ordenación en la concurrencia de recursos. El bloqueo

obedece a un protocolo de adquisición y liberación controlado por medio de la palabra clave ***synchronized*** y funciona únicamente en el contexto de los hilos. Cada instancia de la clase *Object* y sus subclases posee un cerrojo, el cual es utilizado para bloquear a los clientes que intentan invocar un método de un objeto que esta siendo ejecutado por un hilo. La sincronización se puede realizar por bloques o métodos. Ver figura 3.50.

(a)

```
public void setClientInterface(ClientInterface c,
                String strUser) {
  synchronized(htUsers) {
        htUsers.put(strUser, c);
  }
}
```

(b)

```
public synchronized crear( ){
  BufferedWriter outBuffer=null;
  try {
    outBuffer = new BufferedWriter(new FileWriter(path));
  }catch(IOException e){}
}
```

Figura 3.50. (a) Bloque sincronizado. (b) Método sincronizado.

En la sincronización de bloques se toma como argumento el objeto que se bloquea. Esto permite que cualquier método pueda bloquear cualquier objeto. Es decir, cuando un hilo adquiera el cerrojo la ejecución del método o bloque sincronizado será serial, por lo que ningún otro hilo podrá ejecutarlo hasta que sea liberado (ver figura 3.51). La primera vez que un hilo acceda al valor del atributo de un objeto, obtendrá el valor inicial del atributo, o bien el valor que escribió algún otro hilo.

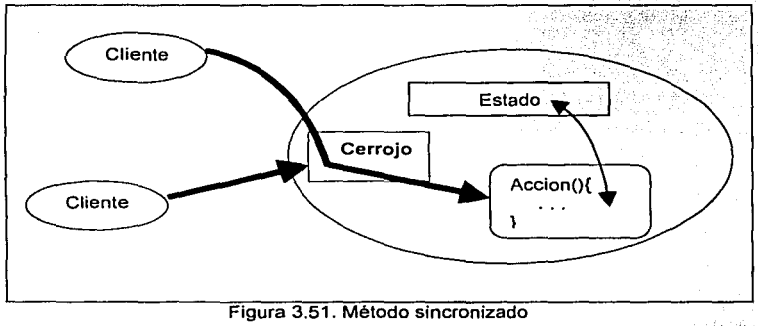


Figura 3.51. Método sincronizado

Cuando un hilo libere un cerrojo, otro hilo podrá adquirirlo (quizás el mismo hilo si ejecuta otro método *synchronized*). La liberación de un cerrojo fuerza a una limpieza de todo lo escrito en la memoria de trabajo empleada por el hilo, y la adquisición de un cerrojo fuerza a recargar los valores de los atributos accesibles. Mientras que las acciones del cerrojo proporcionan exclusión solamente en las operaciones realizadas dentro de un método o de un bloque *synchronized,* los efectos sobre la memoria se definen para cubrir todos los atributos usados por el hilo que realiza la acción.

Por lo tanto, la palabra *synchronized* trata con los cerrojos que permiten protocolos de sincronización de alto nivel, y también se ocupa del sistema de memoria para guardar la representación del valor sincronizado a través de los hilos. Entonces, el término *synchronized* puede verse como un mecanismo, que indica que un método que funciona en un hilo, está dispuesto a enviar y/o recibir cambios en las variables, a y desde los métodos que se ejecutan en otros hilos. Sin embargo, no existe una garantía sobre cuál de los hilos bloqueados será el

siguiente en adquirir el cerrojo o cuando lo hará. Además, no existe mecanismo alguno para descubrir si un cerrojo fue adquirido por algún hilo.

De lo anterior se puede concluir que la sincronización puede determinar las propiedades de atomicidad, visibilidad y ordenación, ya que todos los cambios realizados en un método o bloque *synchronized* son atómicos y visibles con respecto a otros métodos y bloques *synchronized* que emplean el mismo cerrojo. Además, el procesamiento de los métodos bloques *synchronized* dentro de cualquier hilo se realiza según el orden especificado en el programa y aunque el procesamiento de las sentencias de los bloques se ejecutan fuera de orden, éste no puede influir en otros hilos que emplean la sincronización.

Para evitar conflictos de lectura y escritura en la creación de un documento debido a la concurrencia, se implementó el bloqueo sobre el método *createNewDocument*. La figura 3.51 muestra parte del código que crea un documento en la parte del servidor. El método tiene como argumento un objeto de tipo *fileObj*, el cual contiene la información del nombre del documento y de los fragmentos que lo constituyen. Primero, se valida que el colaborador sea miembro del grupo en donde se desea crear el documento (figura 3.52, línea 8). Posteriormente se realiza la inserción del nombre del documento y de los fragmentos en las respectivas tablas *archivos* y *fragmentos* de la base de datos de ELXI (figura 3.52, línea 27).

```
1   public synchronized int createNewDocument(fileObj f) throws RemoteException {
2
3    /**Verificación de rol y permisos**/
4   String strsql = "SELECT nombre_usuario FROM miembros WHERE nombre_grupo='"+f.getGroup()+"' AND
5           nombre_usuario='"+f.getcreatedBy()+"'";
6
7   if(!gLeerBD.abrirConexionBD()) return 0;
8    if((data = (Hashtable) gLeerBD.getDatos(strsql)) == null) return 0;
9     ...
10   /**Inserción del archivo en la BD**/
11   strsql = "INSERT INTO archivos (nombre_archivo, creadopor) VALUES('"+f.toString()+"','"+ f.getcreatedBy()+"')";
12   if(gLeerBD.inData(strsql)==0) return -3;
13   else {
14      strsql = "SELECT id_archivo FROM archivos WHERE nombre_archivo='"+f.toString()
15      if((data = (Hashtable) gLeerBD.getDatos(strsql)) == null) return 0;
16       if(data.size()==1) {
17            newId = Integer.parseInt((String) data.remove("1"));
18            Vector frags = f.getFragments();
19            fragObj frag;
20            for(int i=0; i<frags.size(); i++) {
21                  frag = (fragObj) frags.elementAt(i);
22
23           /**Inserción de fragmentos en la BD**/
24            strsql = "INSERT INTO fragmentos (id_archivo, id_fragmento, nombre_fragmento, creadopor,
25            establoqueado) VALUES("+newId+", "+frag.getid_fragment()+",'"+frag.toString()+ "', '"+
26            frag.getcreatedBy()+"', false)";
27           if(gLeerBD.inData(strsql)==0) return -3;
28                 }
29           StringBuffer strB = f.getContent();
30           int pos = strB.toString().indexOf("id_file=");
31           strB.insert(pos+9, newId);
32
33          /**Creación del archive en el sistema de archivos***/
34          if(stringToFile(rutaDatos+f.getGroup()+separador+f.toString(), strB.toString())) return newId;
35           ...
36  }
```

Figura 3.52.Método que se ejecuta en el servidor ante la solicitud "Crear un documento".

Para registrar el bloqueo del fragmento se implementó el método *registerLockFrag* que tiene como parámetros el nombre del usuario y el objeto fragmento. Lo primero que se realiza es verificar que efectivamente el colaborador pueda obtener el documento a través del rol de escritor en el fragmento (figura 3.53, línea 10) y además que el fragmento no este bloqueado por otro autor (figura 3.53, línea 21). Posteriormente, se ejecuta la sentencia *sql* de actualización sobre la tabla *fragmento* para registrar el bloqueo (figura 3.53, línea 25). Enseguida se crea el hilo de tipo *lockedThread* encargado de contabilizar el tiempo de bloqueo en la parte del servidor (figura 3.53, línea 29). Este método también es sincronizado para evitar conflictos en la adquisición del bloqueo del fragmento.

```
1   public synchronized int registerLockedFrag(String strUser, fragObj fr) throws RemoteException{
2
3   /**Verifica que tenga permisos de administrador o escritor sobre el fragmento**/
4   String strsql ="SELECT roles.nombre_grupo FROM roles WHERE roles.nombre_grupo= "+
5           "miembros.nombre_grupo AND miembros.nombre_usuario = roles.nombre_usuario AND"+
6           "archivos.id_archivo=" + fr.getid_file() +" AND roles.id_archivo=archivos.id_archivo AND"+
7           "roles.nombre_usuario ="'+ strUser+"'" AND roles.rol_doc=(papel='AF' OR papel='EF')";
8
9    if(!gLeerBD.abrirConexionBD()) return iResult;
10    Hashtable datos = (Hashtable) gLeerBD.getDatos(strsql);
11    if(datos.size()>0){
12     String strGroup = (String)datos.get("1");
13      ...
14     /**revisa que no esté bloqueado el fragmento**/
15      strsql = "SELECT establoqueado FROM fragmentos WHERE id_archivo="+fr.getid_file() +" AND
16      id_fragmento="+fr.getid_fragment();
17
18      datos = (Hashtable) gLeerBD.getDatos(strsql);
19      ...
20      /**bloquea el fragmento**/
21       strsql = "UPDATE fragmentos SET establoqueado='t',bloqueadopor ='"+strUser+"',tiempo_bloqueo="+
22            fr.gettimeLocked()+", tiempo_remoto="+fr.getLocalTimeLocked()+ " WHERE id_archivo="+
23            fr.getid_file() + "AND  id_fragmento="+fr.getid_fragment();
24
25       iResult = gLeerBD.inData(strsql);
26
27       if (iResult>0){
28          /**se crea el hilo de bloqueo del fragmento que contabiliza el tiempo en segundos**/
29          lockedThread newThread = new lockedThread(this, strGroup,(new Integer(fr.getid_file())).intValue(),
30          (new Integer(fr.getid_fragment())).intValue(), (new Long (fr.gettimeLocked())).longValue());
31
32          htLockedThread.put(strGroup+fr.getid_file()+fr.getid_fragment(),newThread);
33          newThread.start();
34        ...
35  }
```

Figura 3.53. Método que se ejecuta en el servidor ante la solicitud "Bloquear un fragmento".

En las figura 3.54 y 3.55 se muestran los métodos *run* y *unlocked* de la clase *lockedThread, u*tilizada para crear el hilo que da seguimiento al tiempo de bloqueo de un fragmento. Una vez que se ejecuta el método *run*, el hilo duerme durante un intervalo que es determinado por el tiempo de bloqueo (figura 3.54, línea 4). Cuando este tiempo finaliza, el hilo se despierta e invoca al método *unLocked*, en donde se llama al método *unLockedFrag* de la clase *ServerElxilmpl* para borrar el bloqueo del fragmento en la base de datos (figura 3.55, línea 3). Posteriormente, se verifica si existe una solicitud en la lista de espera para obtener el bloqueo del fragmento. Si es así, el bloqueo es registrado para el cliente de dicha solicitud y mediante su interfaz se le notifica de esta

operación (figura 3.55, línea 20). Asimismo, se les envía a los clientes que se encuentran en la lista de espera un mensaje de la posición en que se encuentran (figura 3.55, línea 24).

```
1   public void run(){
2   try{
3     /**El hilo duerme durante el periodo de bloqueo**/
4     thisThread.sleep(lTime);
5
6    }catch(InterruptedException e){ }
7    /**El hilo despierta cuando el periodo de bloqueo termina**/
8    unlocked();
9   }
```

Figura 3.54. Método run de la clase lockedThread.

```
1   public void unlocked(){
2    try{ /**borrar el bloqueo en la base de datos**/
3      serverElxi.unLockedFrag(iDoc, iFrag);
4    }catch(Exception e){
5     System.out.println("No hay comunicación con el servidor");
6    }
7    /**Verifica si existe una solicitud de bloqueo en la lista de espera para el fragmento que se ha liberado **/
8    synchronized(serverElxi.htWaitQueue){
9
10    ObjectQueue oq; /** objeto en la lista de espera**/
11    ClientInterface ci;
12    LinkedQueue lq; /** lista de espera, cola**/
13    int iResult=0;
14      ....
15      /**Obtiene interfaz del cliente para notificarle la obtención del bloqueo**/
16      ci =(ClientInterface) oq.getInterface();
17      try{
18          if(ci!=null){
19           ci.messageToClient("Espera respuesta del cliente");
20           iResult=serverElxi.registerLockedFrag(oq);
21          }
22      }catch(RemoteException re){ connected=false; }
23     ...
24      /**Notifica a los clientes en que posición están **/
25       for(Enumeration e = lq.elements(); e.hasMoreElements();){
26         ObjectQueue temp =(ObjectQueue) e.nextElement();
27         try {
28           ci =(ClientInterface) oq.getInterface();
29           ci.messageToClient("Ahora te encuentras en la posición "+i);
30         }catch(RemoteException re){ System.out.println("No se encuentra el cliente");
31  }
```

Figura 3.55. Método unlocked de la clase lockedThread.

Para quitar un bloqueo a través de una solicitud cliente se implementó el método *registerunLockedFrag*, en el cual se verifica que el autor que desea quitar el bloqueo, es la persona quien lo solicitó (figura 3.56, línea 9). Enseguida, se localiza el hilo correspondiente al bloqueo en una tabla de tipo *Hash* para destruirlo (figura 3.56, línea 16). Si el hilo no es encontrado, quiere decir que fue destruido. La única causa por la que fue destruido el hilo bajo estas circunstancias es por falta de energía en la máquina servidor, ya que este objeto sólo reside en memoria. Posteriormente, se ejecuta el método *unLockedFrag*, el cual quita el bloqueo de la base de datos y notifica de esta operación a los colaboradores activos en el documento (figura 3.57, línea 8 y 15).

```
1   public int registerunLockedFrag(String strUser, fragObj fr)throws RemoteException {
2
3   /**Verifica que el colaborador tenga el bloqueo**/
4   String strsql="SELECT nombre_grupo FROM archivos WHERE fragmentos.id_archivo=archivos.id_archivo
5   AND fragmentos.id_archivo="+fr.getid_file()+ " AND fragmentos.id_fragmento="+fr.getid_fragment()+" AND
6   fragmentos.establoqueado=true AND fragmentos.bloqueadopor='"+strUser+"'";
7
8   if(!gLeerBD.abrirConexionBD()) return 0;
9   Hashtable datos = (Hashtable) gLeerBD.getDatos(strsql);
10  if(datos.size()!=0){
11    lockedThread killThread = (lockedThread)htLockedThread.remove((String)datos.remove("1")+
12    fr.getid_file()+ fr.getid_fragment());
13
14  /**Destruye el hilo**/
15  if(killThread!=null){
16    killThread.stop();
17    killThread=null;
18  }
19  /**Quita el bloqueo en la base de datos**/
20  else unLockedFrag((new Integer(fr.getid_file())).intValue(), (new Integer(fr.getid_fragment())).intValue());
21  ...
22  }
```

Figura. 3.56. Método que se ejecuta en el servidor ante la solicitud "Quitar bloqueo de un fragmento".

```
1   public boolean unLockedFrag(int iDoc, int iFrag) throws RemoteException{
2
3   /**Quita el bloqueo de unf ragmento en la tabla fragmentos de la BD**/
4   String strsql= "UPDATE fragmentos SET establoqueado='f',bloqueadopor='',tiempo_bloqueo=null,
5           fecha_bloqueo=null WHERE id_archivo="+iDoc+" AND id_fragmento="+iFrag+";";
6
7     if(!gLeerBD.abrirConexionBD()) return false;
8     int iResult = gLeerBD.inData(strsql);
9     if(iResult!=0){
10
11      strsql="SELECT nombre_grupo FROM archivos WHERE id_archivo="+iDoc;
12      Hashtable datos=gLeerBD.getDatos(strsql);
13
14     if(datos.size()!=0){
15       lockFragNotification(strGroup,iDoc,iFrag)
16     ...
```

Figura 3.57. Método invocado por el método *registerunLockedFrag*.

El código que maneja la lista de espera para obtener el bloqueo sobre un fragmento se muestra en la figura 3.58. La lista de espera ha sido diseñada como una estructura de tipo FIFO, y cada fragmento tiene una. El método *waitQueue* verifica en el objeto *htWaitQueue* que exista una lista de espera para un fragmento (en la figura 3.58, la lista de espera es llamada *lq*). Si es el caso, entonces a la lista de espera se le agrega un objeto de tipo *ObjectQueue*, el cual contiene la información de la solicitud del bloqueo (figura 3.58, línea 7). Si el objeto de tipo *ObjectQueue*, existe en la lista, éste no se vuelve agregar, tan sólo se le notifica al cliente de su posición actual (figura 3.58, línea 16). En caso de que no existiera la lista de espera para un fragmento, entonces se crea y se inserta la solicitud correspondiente (figura 3.58, línea 20).

Se puede observar que en el método *waitQueue* se implementó un bloque *sinchronized* que tiene como argumento el objeto *htWaitQueue*, de esta forma se controla que exista solamente una lista de espera para el bloqueo de un fragmento, así como la posición de cada solicitud.

```
1   public String waitQueue (ObjectQueue oq) throws RemoteException{
2
3     synchronized(htWaitQueue){
4
5       /**verifica si existe una lista de espera para un fragmento en particular**/
6       if(htWaitQueue.containsKey(oq.getFile()+","+oq.getFrag())){
7        LinkedQueue lq =(LinkedQueue)htWaitQueue.get(oq.getFile()+","+oq.getFrag());
8
9        for(Enumeration e = lq.elements(); e.hasMoreElements();){
10        ObjectQueue temp =(ObjectQueue) e.nextElement();
11        if(temp.getUser().equals(oq.getUser())&&temp.getFile().equals(oq.getFile()) &&
12         temp.getFrag().equals(oq.getFrag()))  message ="Ya tienes el lugar"+i;
13          i++;
14       }
15       lq.add(oq);                          /**agrega la solicitud a la lista de espera**/
16       message ="Estas en la posición"+lq.size() ;      /**indica cual es su posición**/
17      }
18      else{                                 /**crea la lista de espera **/
19       LinkedQueue lq = new LinkedQueue();
20       lq.add(oq);
21       htWaitQueue.put(oq.getFile()+","+oq.getFrag(),lq);
22       message ="Estas en la posición"+lq.size();       /**indica cual es su posición**/
23       ...
```

Figura 3.58 Método que implementa la lista de espera.

Para concluir con la sección de implementación, en la tabla de la figura 3.59 se han integrado los diferentes roles y permisos que puede adquirir un colaborador en ELXI. Se puede observar que se manejan principalmente tres entidades: grupo, documento y fragmento. En cada una de ellas existen diferentes actores, los cuales van a poder realizar las acciones especificadas por su roles.

Entidad	Roles	Permisos	Observaciones
Grupo	Creador de grupo (C)	•Agregar usuarios de ELXI al grupo. •Borrar a colaboradores del grupo en el caso de que no estén asociados a un documento.	Cualquier usuario de ELXI puede crear un grupo y por omisión se convierte en miembro
Documento	Administrador de documento (AD)	•Otorgar permisos de documento a los colaboradores que pertenecen al grupo. •Borrar el documento mientras no tenga asociado algún colaborador distinto a él.	Cualquier colaborador que pertenezca a un grupo puede crear un documento y por omisión adquiere el papel de administrador de documento.
	Autor (ED ó LD)	•Obtener documento. •Agregar fragmentos. Incluye el permiso de obtener documento. •Sin permisos. El usuario no tiene derecho de obtener el documento aunque pertenezca al grupo.	Cualquier colaborador que tenga el permiso de "obtener el documento", por omisión tiene el derecho de lectura en todos los fragmentos
Fragmento	Administrador de fragmento (AF)	•Un usuario con este derecho puede asignar permisos de fragmento a los usuarios del documento. •Borrar fragmentos sino se encuentra bloqueado.	Cualquier autor que tenga el permiso de agregar fragmentos puede ser administrador del fragmento que inserte.
	Escritor (EF)	•Un usuario con este derecho puede editar el fragmento.	
	Lector (LF)	•Un usuario con este derecho sólo puede leer el fragmento.	

Figura 3.59. Tabla "Roles y Permisos".

Hay que destacar que un colaborador puede adquirir diferentes roles, puede ser creador de grupo, administrador de documento y administrador de fragmento. Para almacenar los roles de cada colaborador, en la tabla *fragmentos* de la base de datos de ELXI, se maneja la nomenclatura que se especifica en los paréntesis que aparecen en la columna *roles* de la tabla de la figura 3.54.

3.23 Resumen

El mecanismo de control de concurrencia se basa principalmente en:

1. La estructura del documento, en el cual se define al fragmento como unidad mínima de concurrencia.
2. Las responsabilidades de los usuarios sobre los documentos, manejadas como roles y permisos almacenados en una base de datos.
3. El control de la edición concurrente en un fragmento mediante el bloqueo.

Con respecto al primer punto se definió, mediante la DTD, cuales son las etiquetas que se manejan para distinguir a los fragmentos en un documento XML. De tal manera que los clientes trabajen con objetos de tipo *fragObj* para las operaciones de crear, actualizar y borrar. En lo que respecta al tamaño del fragmento, los propios colaboradores tendrán que discutir su tamaño y si representa un párrafo, una sección o un capítulo, etc.
A fin de que se establecieran responsabilidades a los colaboradores de ELXI, fue necesario establecer un nueva política, en la que un colaborador sólo puede crear un documento si es miembro de un grupo. Entonces, un grupo tiene como elementos colaboradores y documentos. Sin embargo, esto no quiere decir que cualquier miembro puede leer o escribir en un documento, esto depende de los roles y permisos.
La creación de roles se determinó a través de tres entidades: grupo, documento y fragmento. En la entidad grupo, un colaborador puede adquirir el rol de creador de grupo (si ha creado un grupo) o miembro. En el caso de la entidad documento, un colaborador puede tener el rol de administrador de documento (si ha creado un documento) o autor de documento. Por último la entidad fragmento, los colaboradores pueden adquirir el rol de administrador de fragmento (si ha creado un fragmento u otro administrador le concedió ese rol), escritor de fragmento y lector de fragmento. Asociados a los roles están los permisos, los cuales establecen que acciones son permisibles. Es importante destacar que existen excepciones en las acciones de borrado para cualquier entidad, esto es debido a que se consideró más importante preservar las aportaciones de los autores.
Para resolver el problema en el que dos autores escriben al mismo tiempo en un mismo fragmento, se optó por implementar el mecanismo de bloqueo, debido a que tenemos un ambiente donde existen varios escritores. Para no rechazar las solicitudes de bloqueo sobre un fragmento que ya esta bloqueado se implementó una lista de espera de tipo FIFO, en donde la primera solicitud en llegar es la primera que se atiende.
Para conocer cual ha sido la respuesta del usuario final, al usar este mecanismo en ELXI, se realizaron pruebas de funcionalidad y desempeño que en el siguiente capítulo se analizan.

Capítulo 4

Evaluación y resultados

Una parte importante del ciclo de vida de un sistema es su fase de pruebas. Esta sirve para verificar la funcionalidad o conocer el desempeño del sistema. Es por ello que al incorporar el mecanismo de control de concurrencia en ELXI, también se decidió realizar una evaluación del mismo.

En la primera parte de este capítulo se explica cuál es el entorno gráfico del proceso cliente de ELXI, con el propósito de identificar las diferente áreas que constituyen el espacio de trabajo en la sesión de un colaborador.

Posteriormente, se lleva a cabo la evaluación del sistema ELXI, la cual se divide en dos tipos de pruebas. La primera es la prueba de usuario final, la cual consiste en que un grupo de colaboradores utilice el sistema ELXI para la elaboración de un documento. Siendo el objetivo principal verificar la funcionalidad de los procesos que ejecutan la creación de grupos, la incorporación de miembros, la asignación de roles de documento y fragmento, así como el bloqueo de un fragmento. Este tipo de prueba se realizó gracias a la participación de un grupo de colaboradores, miembros del personal del Instituto Mexicano del Petróleo. El seguimiento que se le dio a esta evaluación fue la observación de las acciones de los colaboradores, así como el registro de sus comentarios y la obtención de una calificación del mecanismo de concurrencia a través de un cuestionario.

La segunda prueba de evaluación es la de desempeño. En esta se expone que tan eficiente es el almacenamiento de la información contenida en los fragmentos. Para realizar este estudio se compararon dos maneras de almacenar los datos. La primera establece que los fragmentos de un documento sean almacenados en un mismo archivo. Mientras que la segundo forma, propone que cada fragmento de un documento sea almacenado en un archivo.

En cada una de las pruebas se despliegan resultados que son útiles para comprobar, así como mejorar la funcionalidad del sistema ELXI.

4.1 Espacio de trabajo

El proceso cliente es una aplicación desarrollada en JAVA, la cual define un espacio de trabajo. Este esta constituido por diferentes componentes: menú desplegable, escritorio, árbol de documentos, ventana de mensajes y un editor (ver figura 4.1).
El escritorio de trabajo es un área en donde se despliegan diferentes ventanas, por ejemplo la de creación de usuarios, creación de grupo, agregar usuarios a grupos, borrar usuario de un grupo, incluyendo la ventana del editor.
El árbol de documentos muestra la estructura de archivos para cada colaborador en particular. Una carpeta representa el grupo al que pertenece y los archivos contenidos en ella son los documentos a los que tiene acceso. Mediante la selección de uno de estos elementos se puede crear un grupo, un documento, o bien abrirlos.

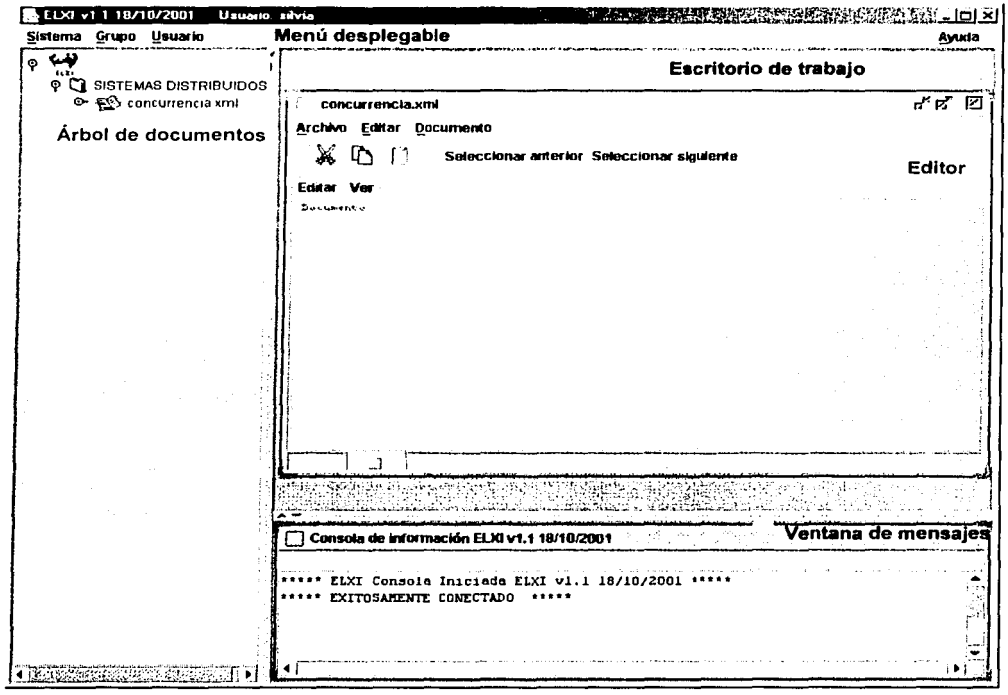


Figura 4.1 Interfaz del cliente ELXI.

En la venta de mensajes de la figura 4.1, se despliegan avisos del funcionamiento del proceso cliente, del servidor, o bien de otros clientes.
Finalmente, el editor de ELXI sólo puede llevar a cabo las operaciones básicas de edición: pegado, copiado y guardar documento. Hasta el momento solo se ha trabajado en el almacenamiento de la información y no en el formato del documento.

4.2 Pruebas de usuario final

Las pruebas de usuario final tienen como propósito saber si el mecanismo de control de concurrencia es funcional en la elaboración de documentos de manera colaborativa. Estas pruebas se llevaron a cabo gracias a la participación de un grupo de colaboradores, miembros del personal del Instituto Mexicano del Petróleo. Desde un inicio, los colaboradores mostraron un gran interés en conocer ELXI y en el hecho de trabajar en equipo para la elaboración de un tipo de documento. Antes que nada, los autores recibieron algunas horas de capacitación para usar y familiarizarse con el sistema ELXI. Se estableció que los colaboradores trabajarían en la producción de un documento de tipo ***Norma***. En todos los sitios donde los colaboradores trabajaron se instaló el sistema ELXI. Antes de arrancar la edición colaborativa, los autores se reunieron frente a frente para decidir quien sería el creador del grupo, el administrador del documento y que roles tomarían el resto de los autores.

Una vez definidos los roles, el primer paso fue incorporar al grupo de colaboradores como usuarios del sistema ELXI. Esta tarea la realizó el administrador[42] de ELXI, quien se encargó de registrar sus nombres e información adicional desde su sesión. En la figura 4.2. se muestra la ventana *usuarios del sistema*, en la cual se despliega una lista en la que se indica el nombre completo del colaborador, el correo electrónico y el estado que se encuentra cada proceso cliente (conectado o no al servidor).

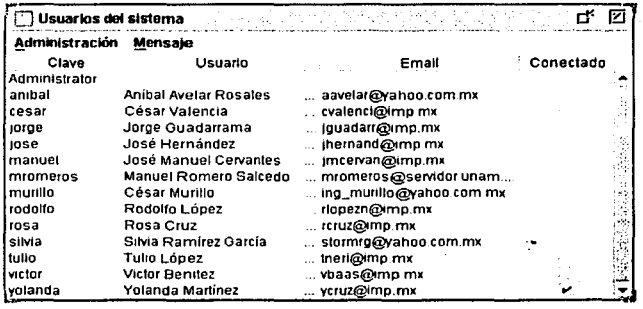


Figura 4.2. Ventana "Usuarios del sistema ELXI".

Como anteriormente se mencionó, antes que un colaborador pueda crear un documento debe ser miembro de un grupo, entonces primero se debe crear un grupo. En la figura 4.3 se listan los grupos existentes en ELXI, de los cuales nos centraremos únicamente en las acciones del grupo NORMA.

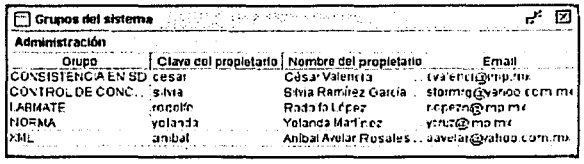


Figura 4.3. Ventana "Grupos del sistema ELXI".

[42] El usuario Administrador es registrado en la base de datos cuando se inicia el proceso servidor por primera vez, por lo que siempre existe en el sistema ELXI.

Ahora corresponde agregar los miembros al grupo *NORMA*. En la figura 4.4 se muestra la lista de los miembros del grupo y asociados a ellos se especifica su correo electrónico y el estado del proceso cliente (conectado o no al servidor).

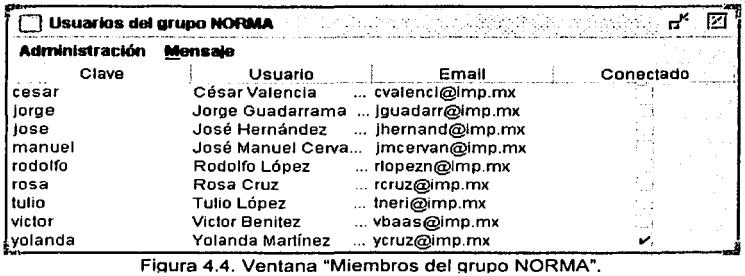


Figura 4.4. Ventana "Miembros del grupo NORMA".

En la figura 4.5 se muestra en el árbol de documentos una carpeta llamada "NORMA", la cual representa el grupo al que el usuario pertenece. Al darle un clic con el botón derecho del ratón, aparecerá un menú desplegable con la opción de crear documento.

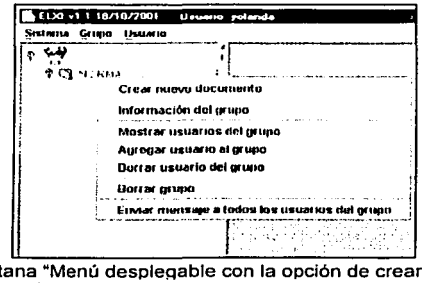


Figura 4.5. Ventana "Menú desplegable con la opción de crear un documento".

Una vez seleccionada la opción *Crear documento*, del menú desplegable de la figura 4.5, aparecerá un cuadro de diálogo en el cual se pedirá el nombre, el tipo y el número de fragmentos que contendrá el documento en su fase inicial y el tipo de documento a crear (ver figura 4.6).

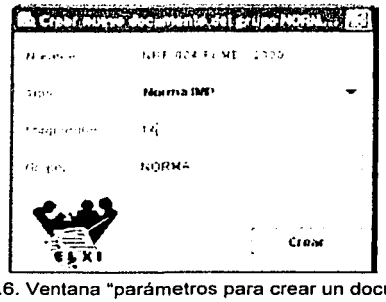


Figura 4.6. Ventana "parámetros para crear un documento".

En la figura 4.7 se muestra el documento *NFR-004-PEMEX-2000.xml* creado por el colaborador *Yolanda*. Ahora en el árbol de documentos se visualizan cuatro elementos: el icono de ELXI que representa la raíz de los documentos, la carpeta que representan el grupo al que pertenece el colaborador, el documento cuyo icono es un libro y el fragmento representado por una hoja. La carpeta con el nombre de *NORMA* contiene dos documentos *NFR-004-PEMEX-2000.xml* y *comentarios2.xml*, cada uno constituido por diferentes fragmentos.

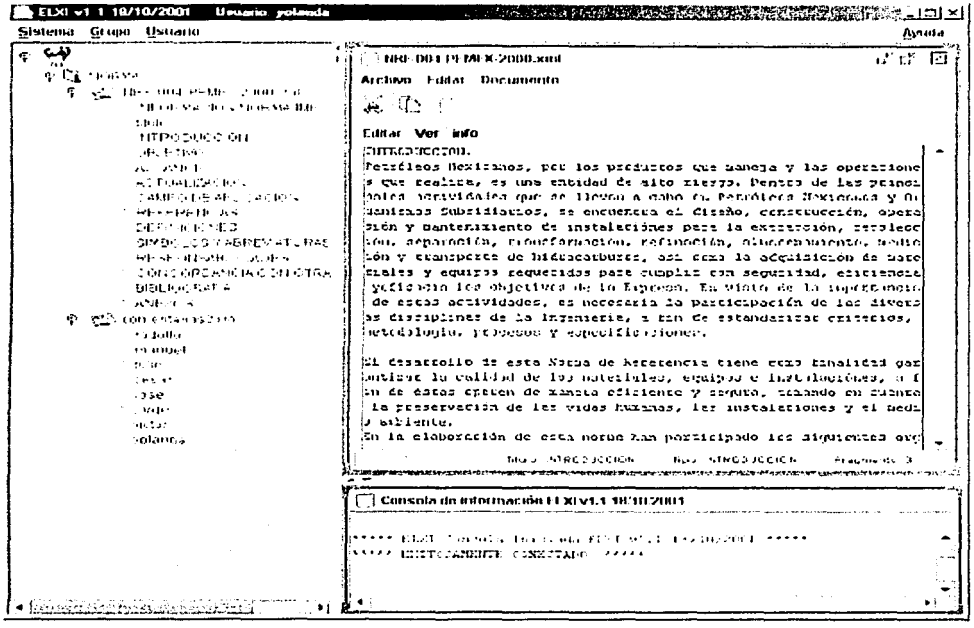


Figura 4.7. Documento *NFR-004-PEMEX-2000.xml*.

Una vez creado el documento, corresponde al administrador del documento asignar los permisos de documento. En la figura 4.8, como el colaborador *Yolanda* es el creador del documento, tiene por omisión el rol de *administrador de documento*. Los demás miembros del grupo han sido seleccionados como autores, con el permiso de obtención de documento.

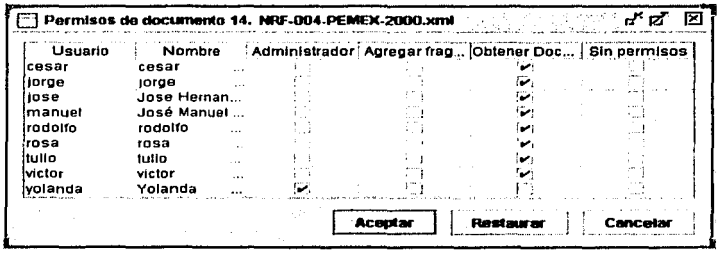


Figura 4.8. Lista de los roles de autores sobre el documento *NFR-004-PEMEX-2000.xml*.

Posteriormente, le corresponde al administrador de cada fragmento asignar los roles de escritor o lector a cada uno de los autores. La figuras 4.9 muestra un ejemplo de la asignación del rol de escritor a los colaboradores cuyos nombres de usuario son jorge, rodolfo, manuel y jose, en el fragmento 3 del documento *NFR-004-PEMEX-2000.xml*.

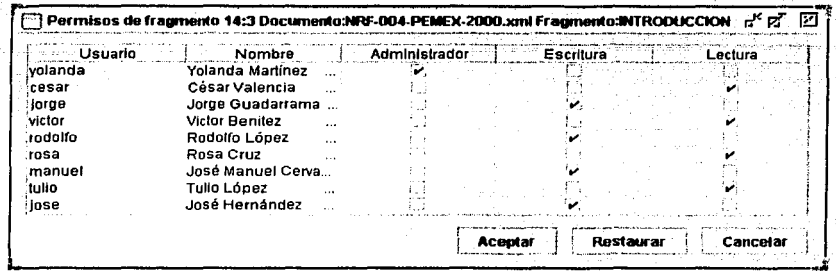


Figura 4.9. Lista de los roles de los autores sobre el fragmento 3 del documento *NFR-004-PEMEX-2000.xml*.

Para bloquear un fragmento en el editor, el autor debe dar un clic en el botón derecho del ratón sobre el fragmento en donde va escribir. Enseguida se despliega un menú emergente, como el que se muestra en la figura 4.10.

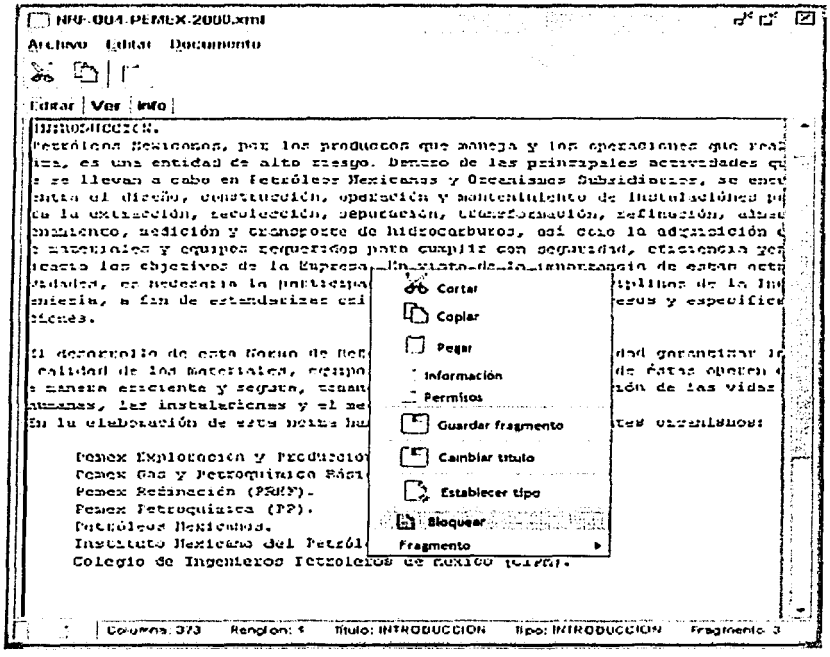


Figura 4.10. Menú emergente del editor.

El menú emergente tiene la opción de bloquear y al darle un clic con el botón izquierdo del ratón aparecerá la ventana de tiempo de bloqueo siempre y cuando el colaborador tenga el rol de escritura en la base de datos. La ventana de tiempo de bloque se muestra en la figura 4.11 y en ella se establece el tiempo en horas o minutos.

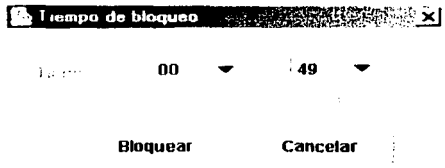


Figura 4.11. Ventana "Tiempo de bloqueo".

Si el autor decidió realizar un bloque de más de 10 minutos, el proceso cliente despliega el mensaje de la figura 4.12. Este mensaje le sirve al proceso cliente para saber si el escritor desea ser avisado acerca de la finalización de su bloqueo. En el caso de que la respuesta sea afirmativa, al escritor le recordará que debe guardar sus cambios antes de que el proceso servidor le quite la escritura activa sobre el fragmento. Es importante hacer notar que si no se tiene el bloqueo del fragmento no se podrán guardar las modificaciones hechas en él.

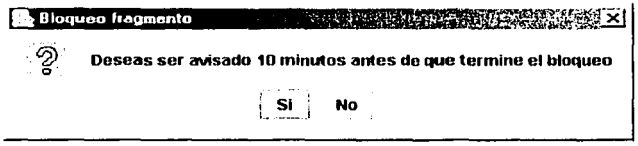


Figura 4.12. Mensaje en donde se determina si el autor desea ser avisado cuando termine su bloqueo.

Para enterar al autor de que ha obtenido el bloqueo del fragmento, se incorporaron ciertos indicaciones en la interfaz gráfica. Primero, la caja de texto que contiene el fragmento en la ventana del editor será enmarcada de color rojo. Asi también, el color del texto cambia de gris a negro, esto indica que el fragmento esta habilitado para la edición. Cada fragmento tiene asociado un icono que se encuentra en la parte inferior derecha y dependiendo de su color es su estado. Si tiene el color blanco representa que nadie esta trabajando en él, pero si es de color verde indica que el autor ha adquirido el bloqueo y por lo tanto tiene habilitada la acción de escritura en el fragmento solicitado. Por último, si el icono es de color rojo anuncia que alguien más tiene el bloqueo de ese fragmento, por lo tanto no puede solicitar la acción de escritura en él en ese momento. El estado del icono también se puede visualizar en el árbol de documentos (ver figura 4.13).

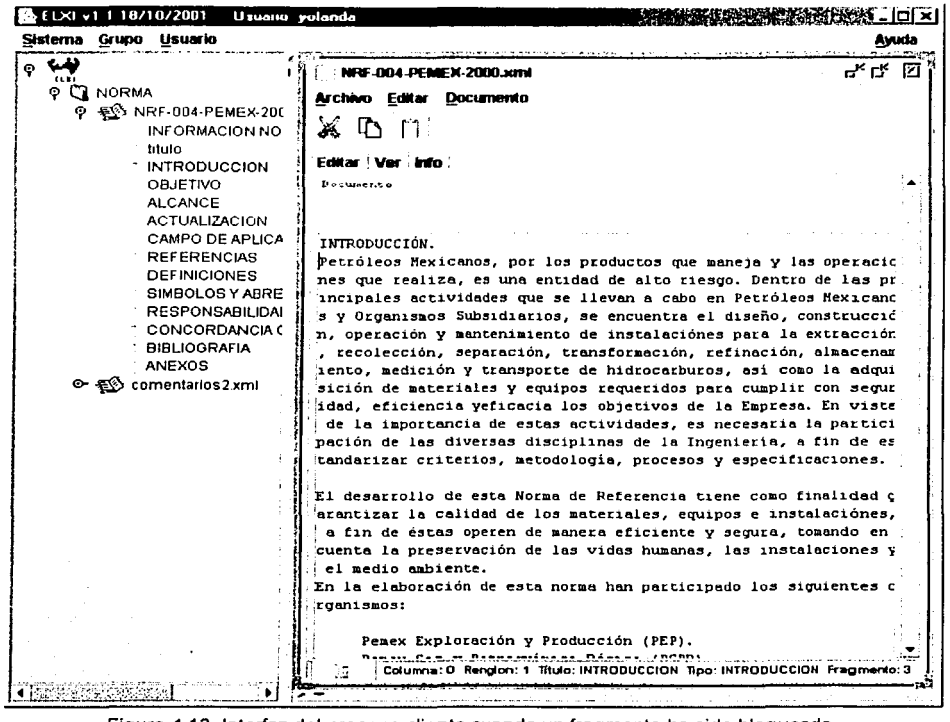


Figura 4.13. Interfaz del proceso cliente cuando un fragmento ha sido bloqueado.

Después de que el grupo *NORMA* terminó de elaborar su documento, se consideró que los colaboradores tenían la experiencia necesaria para evaluar el sistema ELXI a través de un cuestionario. Éste consiste de una serie de reactivos referentes a diferente temas, entre ellos podemos mencionar conciencia de grupo, funcionalidad del editor, concurrencia y replicación de la información. Sin embargo, siendo el principal interés de este trabajo de tesis saber el grado de funcionalidad y aceptación del mecanismo de control de concurrencia, solamente se hará un análisis de las respuestas a las preguntas concernientes al problema mencionado. Estas preguntas se reproducen en la figura 4.14. Sus respuestas fueron computadas, asignando valores a cada una de las siguiente cinco opciones: *Excelente* (valor=5), *Bueno* (valor=4), *Regular* (valor=3), *Deficiente* (valor=2) y *Malo* (valor=1). Como se puede observar el máximo valor que puede tener una respuesta es 5 y la mínima es 1. Los resultados obtenidos del cuestionario de evaluación se muestran en la gráfica de la figura 4.15.

1. ¿Cómo califica la opción de crear grupos en ELXI para identificar a los diferentes equipos de trabajo ?
2. ¿Considera una buena medida que cualquier usuario de ELXI pueda crear un grupo de trabajo?
3. ¿Es una buena medida que los creadores de documentos (administradores de documento) establezcan que miembros del grupo pueden obtener el documento?
4. ¿Cómo califica la división del documento en fragmentos?
5. ¿Cómo califica la existencia de diferentes permisos de documento para controlar la creación del documento?
6. ¿Cómo califica la existencia de diferentes permisos de fragmento para controlar la edición de fragmentos?
7. ¿Qué opina de los diferentes colores que el icono, ubicado en la parte inferior izquierda del editor, adquiere para indicar el estado del fragmento seleccionado? Los estados son BLANCO(disponible para editar), ROJO(no disponible para editar) y VERDE(editando).
8. ¿Cómo considera usted la opción de especificar el tiempo de bloqueo para editar un fragmento?
9. El máximo tiempo de bloqueo es de 23:59 hrs. ¿Considera que es suficiente este tiempo para editar un fragmento?
10. ¿Considera usted una buena opción que el sistema avise 10 min. antes de que se terminar el bloqueo, siempre y cuando lo haya bloqueado más de 10 min.?
11. ¿Cómo considera usted la medida de apartar su turno de bloqueo de un fragmento cuando el fragmento ya esta bloqueado por otra persona?
12. Si ha apartado su turno de bloqueo ¿Cómo califica la forma en que obtiene el bloqueo cuando su turno de edición llega?
13. ¿Cómo califica el proceso de bloquear-editar-guardar, para hacer efectivos sus aportes a un fragmento?

Figura 4.14. Preguntas del cuestionario de evaluación del mecanismo de control de concurrencia

En la figura 4.15 se puede observar la evaluación que realizó el grupo de colaboradores sobre cada una de las partes que conforman el mecanismo de control de concurrencia expuestas en las preguntas del cuestionario. Es posible notar que la respuesta a la pregunta 1 no es tan positiva. Sin embargo, el resultado no se debió a que la forma de crear grupos estuviera funcionando mal, sino que, al incorporar un conjunto de colaboradores a un grupo, era preferible poder seleccionarlos de una lista y agregarlos al mismo tiempo, en lugar de especificar el nombre de cada uno de manera individual. De cualquier forma, la mayoría de los colaboradores opinaron que la creación de los grupos contribuye a la organización del trabajo de edición colaborativa, así como al control de acceso de los documentos.

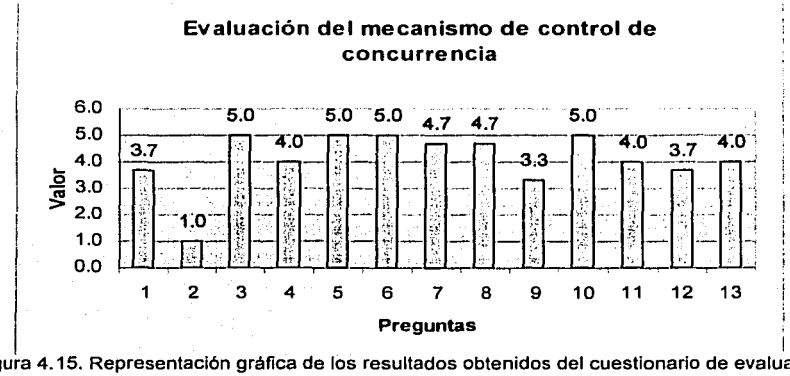


Figura 4.15. Representación gráfica de los resultados obtenidos del cuestionario de evaluación.

Un punto negativo que encontraron los colaboradores, el cual es importante destacar, concierne a las respuestas obtenidas de la pregunta 2. Todos los colaboradores consideraron una mala opción que cualquier usuario de ELXI pudiera crear un grupo de trabajo. Las razones que expusieron fueron que se trataba de un principio que iba en contra de las políticas de seguridad del Instituto. En efecto, todo el personal cuenta con perfiles bien definidos que le permiten manipular y acceder a cierta información del Instituto. Esto con el fin de prevenir robos de información o acceso a la misma sin autorización. Por tanto, los colaboradores opinaron que la seguridad estaba en riesgo si se permitía que cualquier autor de ELXI pudiera crear un grupo de trabajo sin autorización.

Una solución que se propuso a este problema es la de establecer que el administrador del sistema fuera el único quien pudiera crear los grupos. En caso de mayor necesidad de flexibilidad, entonces se podría implementar que el rol de creador de grupo fuera dinámico. Es decir, asignar o remover el rol dependiendo de si se le da la autorización de crear grupo al colaborador.

En las respuestas a la pregunta 4, los colaboradores expusieron que la división de un documento en fragmentos era un nuevo concepto para ellos. No tuvieron problema para asimilarlo, ya que las diferentes partes del documento se identificaban claramente. Sin embargo, opinaron que era importante que el sistema ELXI integrara una herramienta síncrona para llevar a cabo una fase de planeación y/o consolidación que les permitiera discutir y decidir, entre otras cosas, sobre cual sería la mejor división y repartición de fragmentos de un documento.

Debido a la forma en como el grupo de colaboradores organizó su forma de trabajo, se observó que no explotaron al máximo algunas de las características que ofrece el mecanismo de concurrencia. Por ejemplo, en ELXI se establecido que se podía bloquear un fragmento, cuyo tiempo máximo podía ir hasta 23:59 hrs. Esto con el fin de brindar al autor la oportunidad de seguir editando aún cuando se encontrara trabajando en algún otro sitio (que podría ser incluso distante). Sin embargo, en la pregunta 9 se observó que las respuesta que los colaboradores dieron sobre el tiempo de bloqueo fueron que éste era suficiente con 8 hrs., ya que eso duraba la jornada de trabajo y no tenían la necesidad de cambiarse físicamente de lugar para seguir trabajando en su documento.

Todos los colaboradores estuvieron de acuerdo con las medidas expuesta en la pregunta 11 y 12. Sin embargo, hicieron algunos comentarios concernientes a que se deberían considerar prioridades en las solicitudes de los colaboradores. De tal manera que cuando llegue una solicitud de carácter urgente se lleve a cabo antes que las otras que se encuentran en lista de espera.

En lo que respecta al resto de las preguntas en esta temática, tuvieron una respuesta muy positiva. En particular, las respuestas de la pregunta 10, sobre el aviso del fin de bloqueo, indicaron que fue muy bueno disponer con esta opción. Asimismo, estuvieron de acuerdo en que los administradores de documento deben ser los únicos en definir que miembros del grupo pueden recuperar una copia del documento (pregunta 5) y en los fragmentos (pregunta 6) como una forma de asignación de responsabilidades a los colaboradores del sistema.

Aunque el concepto de fragmentos de documento era nuevo para ellos, el proceso de bloquear, editar y guardar las modificaciones fue calificado como bueno, ya que comprobaron que realmente se controla la edición concurrente de una sección del documento.

Por otro lado, opinaron que el administrador de documento debería contar con mayores facultades, como podría ser la de controlar totalmente las operaciones que se pueden hacer sobre los documentos (ej. borrar documentos, quitar bloqueos cuando el autor está escribiendo, modificar los roles de los autores en los fragmentos, etc.). Estas opiniones fueron muy interesantes. Sin

embargo, después de un trabajo de reflexión, se pensó que la implementación de estas opiniones podría acarrear consecuencias y penalizaciones durante el trabajo de edición colaborativa (ej. no disponibilidad de la información, pérdidas de información, etc.). Gracias a los roles de los colaboradores, es posible controlar toda acción de escritura sobre los fragmentos de un documento. Un aspecto interesante es que los roles de un administrador, ya sea de fragmento o de documento, no lo habilita para borrar la información que algún autor haya incorporado en un fragmento. La idea de todo esto es que no se pierda ninguna contribución, ni ningún cambio, sino hasta que sea el mismo colaborador responsable de su texto, quien lo decida.
Dentro de las opiniones que se recabaron del grupo de autores, se sugirió incorporar en el sistema ELXI una herramienta que permitiera configurar la asignación de responsabilidades a los colaboradores del sistema. Aquí se propone que en cada grupo se elabore un diagrama de jerarquía, que permita especificar si existirá una máxima autoridad y quienes serán los colaboradores subordinados. Asociado a cada elemento de este diagrama, se especificarían las actividades que el colaborador en el documento puede desempeñar.
Al igual que la manera de determinar los permisos de los colaboradores en el documento y los permisos de fragmento (pregunta 5). Sin embargo, opinaron que el administrador de documento debería tener la facultad de tener el control total del documento, es decir que tuviera la facultad de borrar el documento, quitar bloqueos aun cuando el autor esta escribiendo y quitar derechos de fragmentos a autores aunque no sea administrador de fragmento. Si se implementan esta observaciones, algunas de las consecuencias y penalizaciones serían la no disponibilidad de la información que tendría un autor a pesar de las contribuciones realizadas al documento, perdidas de documentos, así como de actualizaciones realizadas por los autores.
Como se pudo observar el grupo de colaboradores realizaron algunas solicitudes de acuerdo a sus necesidades o jerarquía de trabajo pero para implementarlas hay que cuidar un aspecto principal de ELXI, la flexibilidad, es decir que su configuración se pueda adecuar lo más posible a la forma de trabajo de la institución o empresa donde se utilice.

4.3 Pruebas de desempeño

Las pruebas de desempeño que se han realizado corresponden al almacenamiento de la información en un medio concurrente que es el archivo. Cuando se propuso la estructura del documento, se estableció que éste fuera divido en fragmentos. En la parte del diseño se observó que los fragmentos de un documento son almacenados en un archivo. Sin embargo, surge la inquietud de saber que tan eficiente resulta esta estrategia implementada. Para tener un punto de comparación, en esta sección se proponen trabajar con dos esquemas buscando minimizar el tiempo de respuestas a las solicitudes de los clientes.
El primer esquema es manejar un documento como un archivo, constituido por secciones llamadas fragmentos. A este esquema se le ha nombrado **documento-archivo** y el diagrama de actividad para crear el documento se muestra en la figura 4.16.a. Consiste en crear un objeto documento y posteriormente los objetos fragmentos (el número de éstos es indicado por el creador del documento). Enseguida, un objeto fragmento es agregado al objeto documento y si existen más fragmentos se vuelve a ejecutar la operación *insertar_fragmento_documento*. Cuando ya no existen más fragmentos que agregar, se crea un archivo en el sistema de archivos, al cual se le

inserta el contenido del objeto documento. Finalmente, el archivo es guardado. Este proceso es el que se ha implementado en ELXI, al incorporar el mecanismo de concurrencia.

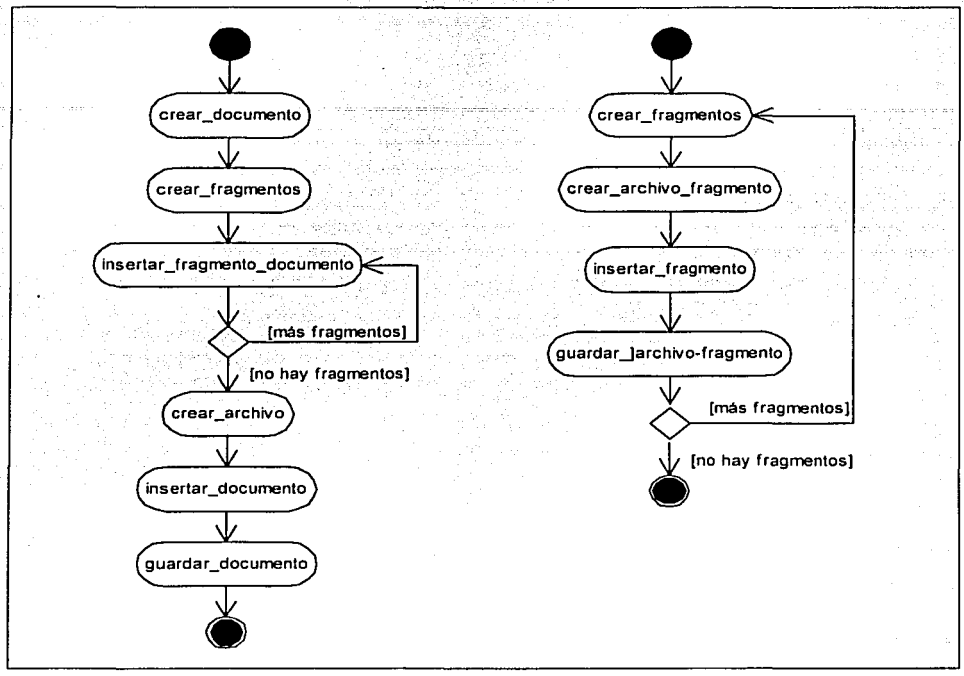


Figura 4.16. Diagramas de actividad. (a) Creación de un documento en un archivo (b) Creación de un documento en varios archivos, cada uno de ellos contiene un fragmento.

El segundo esquema mostrado en la figura 4.16.b, propone que el contenido de cada fragmento sea almacenado en un archivo y la conjunción de ellos formen un documento. Es decir, un documento esta constituido por un conjunto de archivos y no por uno como en el esquema anterior. Este segundo esquema ha sido nombrado como **fragmento-archivo** y consiste en crear únicamente objetos fragmentos, de tal manera que cada uno de estos objetos sea almacenado en un archivo. Esta actividad se repite dependiendo del número de fragmentos a crear.

Para conocer la eficiencia de ambos esquemas se midió el tiempo que dura el proceso de almacenamiento. La manera que se lleva a cabo el almacenamiento del contenido de un fragmento en un **documento-archivo** se explicó en el diagrama de secuencia de la figura 3.30 (pag. 78). Para el caso del almacenamiento del contenido de un fragmento en un **fragmento-archivo** consiste en localizar el archivo (cuyo nombre es el del fragmento). Posteriormente será analizado sintácticamente por el objeto de tipo *Parser*, quien se encargará de verificar que el archivo sea un documento XML válido. Enseguida, el objeto *EditDocumento* crea el objeto documento con su respectivo objeto fragmento a partir del archivo. Después, se localiza el objeto fragmento mediante su número de identificación para sobrescribir su contenido. Finalmente, el objeto fragmento es guardado en el archivo respectivo. Ver figura 4.17.

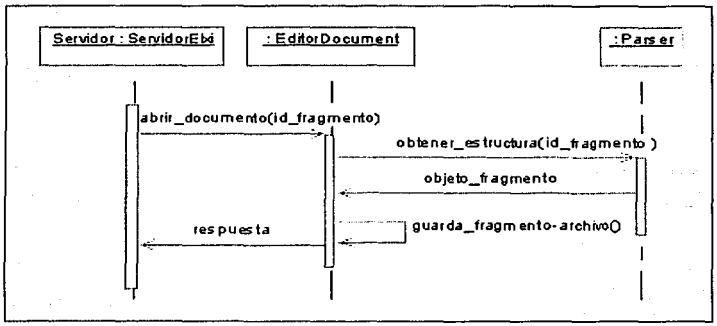


Figura 4.17 Diagrama de secuencia "Almacenamiento del contenido de un fragmento en un fragmento-archivo".

Cada vez que se guarda un fragmento, el documento es bloqueado por el sistema de archivos. De esta manera, cada hilo que realice la operación de escritura debe esperar al hilo que posee el recurso (documento). Por lo tanto, si un conjunto de hilos solicitan guardar un fragmento en un mismo documento y al mismo tiempo, entonces el tiempo de almacenamiento de todos los hilos es igual al tiempo que duró el último hilo en escribir. Esto se representa en la siguiente expresión:

$$T = Dif(T_f, T_i)_{Hk}$$

donde T=Tiempo de duración máximo, T_i=Tiempo inicial de ejecución del hilo, T_f=Tiempo final de ejecución del hilo, H = Hilo y k>0. La operación *Dif* es la diferencia entre T_f y T_i del último hilo H_k en terminar su ejecución.

En la figura 4.18 se muestra una tabla de datos estadísticos para el esquema **documento-archivo**. Esta tabla esta constituida por seis casos. En cada uno de ellos se ejecutaron al mismo tiempo un número de hilos (simulación de procesos clientes), los cuales trabajaron con un fragmento diferente sobre un documento. Cada caso está constituido por diez muestras en las que se realizó la medición del tiempo máximo que dura el proceso de almacenamiento de los fragmentos.

	No. colaboradores	M1 [mseg]	M2 [mseg]	M3 [mseg]	M4 [mseg]	M5 [mseg]	M6 [mseg]	M7 [mseg]	M8 [mseg]	M9 [mseg]	M10 [mseg]	Prom [mseg]	Prom [seg]
1	5 colaboradores	1650	1718	1672	1720	1688	1631	1681	1708	1680	1662	1681	1.6810
2	10 colaboradores	2217	2212	2183	2279	2232	2286	2191	2286	2219	2173	2228	2.2278
3	15 colaboradores	2480	2540	2407	2513	2560	2461	2464	2482	2589	2539	2504	2.5035
4	20 colaboradores	3062	3087	2681	3014	3073	2995	2654	2651	3076	2861	2915	2.9154
5	25 colaboradores	3211	3320	3252	3247	3307	3138	3198	3190	3300	3286	3245	3.2449
6	30 colaboradores	3476	3464	3443	3371	3492	3525	3495	3465	3376	3334	3444	3.4441

Figura 4.18. Datos estadísticos del tiempo de duración del almacenamiento de la info. de fragmentos en un archivo.

Para cada caso de la tabla de la figura 4.18, las muestras son promediadas y sus resultados serán utilizados en el siguiente análisis. Se puede observar que para cinco colaboradores el máximo tiempo que duró un hilo en el proceso de almacenamiento es de 1.6810 seg. En los siguientes casos se observa que este tiempo va aumentando de acuerdo al número de colaboradores. Para treinta colaboradores se tiene un tiempo de 3.4441 seg. siendo más del doble de tiempo que para

el primer caso. Este resultado es de esperarse, ya que cada solicitud cliente debe aguardar a que se libere el archivo en el que se almacenará el contenido del fragmento.
En la figura 4.19 se muestra una tabla de datos estadísticos para el esquema **fragmento-archivo**. Igualmente esta constituido por seis casos. Sin embargo, el tiempo de duración del proceso de almacenamiento se refiere al tiempo de duración máximo de un hilo, en un ambiente donde existen otros hilos que desean almacenar información en diferentes archivos. Así que el tiempo de duración máxima, en un ambiente donde se ejecuta un grupo hilos que trabajan en diferentes archivos, está dado por la siguiente expresión:

$$T= T_{max}(Dif\ (T_f, T_i)_{H1}, \ldots, Dif(T_f, T_i)_{Hk})$$

donde T_{max}=tiempo de duración máximo, T_i =tiempo inicial de ejecución del hilo, T_f =tiempo final de ejecución del hilo, H = Hilo y k>0. La operación *Dif* es la diferencia entre T_f y T_i de un hilo H_k.
En el caso de 5 colaboradores el máximo tiempo de ejecución de un hilo es de 1.5040 seg. Para los demás casos, también se observa que el tiempo se incrementa cuando el número de colaboradores aumenta. Este resultado es de esperarse, ya que la carga de trabajo para el proceso servidor es mayor.

	No. colaboradores	M1 [mseg]	M2 [mseg]	M3 [mseg]	M4 [mseg]	M5 [mseg]	M6 [mseg]	M7 [mseg]	M8 [mseg]	M9 [mseg]	M10 [mseg]	Prom [mseg]	Prom [seg]
1	5 colaboradores	1504	1468	1472	1532	1493	1484	1474	1543	1538	1532	1504	1.504
2	10 colaboradores	1668	1633	1656	1656	1679	1621	1737	1662	1806	1639	1676	1.676
3	15 colaboradores	1793	1796	1802	1763	1763	1767	1770	1750	1801	1822	1783	1.783
4	20 colaboradores	2046	2111	2060	2099	2061	2082	2027	2052	2203	2073	2081	2.081
5	25 colaboradores	2249	2191	2229	2219	2239	2130	2210	2252	2191	2241	2215	2.215
6	30 colaboradores	2630	2475	2414	2330	2314	2421	2414	2410	2620	2350	2438	2.438

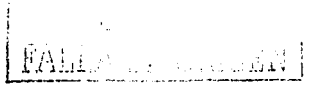
Figura 4.19. Datos estadísticos del tiempo de duración del almacenamiento de la info. de fragmentos en archivos.

En la tabla de la figura 4.20 se puede observar que para los diferentes esquemas, la duración del proceso de almacenamiento es mayor en el esquema **documento-archivo** que en el esquema **fragmento-archivo.**

No. colaboradores	documento-archivo	fragmento- archivo	Diferencia [mseg]	Diferencia [seg]
5 colaboradores	1681	1504	177	0.177
10 colaboradores	2228	1676	552	0.552
15 colaboradores	2504	1783	721	0.721
20 colaboradores	2915	2081	834	0.834
25 colaboradores	3245	2215	1030	1.030
30 colaboradores	3444	2438	1006	1.006

Figura 4.20. Comparación de tiempos entre los esquemas documento-archivo y fragmento-archivo.

En el primer caso, donde existen únicamente cinco colaboradores, el tiempo de almacenamiento para el esquema **documento-archivo** es de 1.681 seg. Esto significa que son 0.177 seg. más que en el de **fragmento-archivo**. Para 10 colaboradores el proceso del **documento-archivo** dura 1.006 seg. más que en el de **fragmento-archivo**. Es posible notar que a medida que el número de colaboradores aumenta, el tiempo de almacenamiento crece y la diferencia de tiempos entre los esquemas también. Con 30 colaboradores, hay una diferencia de 1.0006 seg. entre los procesos de ambos esquemas. Esta diferencia es mayor que para el primer caso.



En las gráficas de las figuras 4.21 a y b se representan los datos de la tabla de la figura 4.20. La gráfica de la figura 4.21.a representa el tiempo en milisegundos segundos.

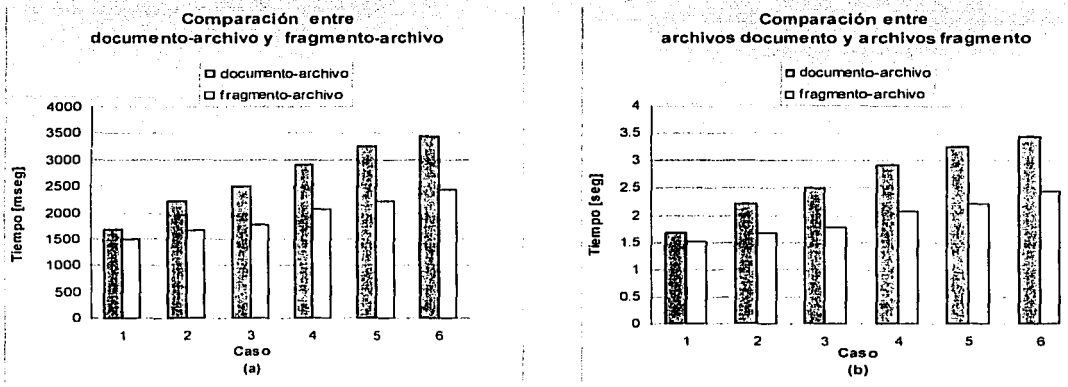


Figura 4.19. Representación gráfica de los resultados obtenidos de la tabla de la figura 4.17

De lo anteriormente expuesto, el esquema **fragmento-archivo** registró tiempos menores en el proceso de almacenamiento, debido a que las diferentes solicitudes clientes accedieron a diferentes archivos al momento de guardar el contenido de un fragmento. Por el contrario, el esquema **documento-archivo** registró tiempos mayores debido a que cada solicitud cliente tenía que esperar hasta que el archivo fuera liberado por otro cliente. Sin embargo, la diferencia de tiempo que existe entre los esquemas para cada caso no excede de más de 2 seg, por lo que se recomienda continuar con el mismo procedimiento de almacenamiento y estructura del documento en un archivo. Pero en caso de que se desee recuperar esos segundos que ofrece el esquema fragmento-archivo, entonces habría que considerar una reestructuración en el sistema de archivos. Sería conveniente que dentro del directorio en el que se define el grupo de colaboración, existiera otro directorio cuyo nombre fuera el nombre del archivo, el cual contendría los archivos referentes a cada fragmento. Esto se implementaría tanto en la parte cliente como en la servidor. Asimismo, es importante considerar la forma en que se construiría el documento para visualizarlo y la manera en que se borrarían y actualizarían los fragmentos en los archivos.

4.4 Resumen

Los resultados de las pruebas de usuario muestran que el mecanismo de control de concurrencia en ELXI no presentó un mal funcionamiento y de las siguientes medidas que se implementaron tuvieron una buena aceptación:

- Organización del trabajo a través de grupos.
- Estructuración de documentos en fragmentos.
- Asignación de roles a los colaboradores sobre documentos y fragmentos.
- El proceso de bloqueo.
- Visualización del estado del fragmento.
- Reservación del bloqueo.

Sin embargo, hubo varias comentarios para complementar la aplicación y mejorarla, las cuales se resumen en los siguientes puntos:

- Mejorar el proceso de agregar miembros a un grupo.
- Incorporar el permiso de crear grupo a los colaboradores, asignado por un administrador.
- Incluir una herramienta de planeación.
- Permitir al administrador del documento cambiar las solicitudes de los colaboradores en la reservación del bloqueo.
- Permitir la operación de borrado de documento o fragmento al administrador del documento sin considerar la contribución de otros autores y la opinión del administrador del fragmento.

Actualmente, los puntos 1 y 2, se están considerando en una nueva fase del desarrollo de ELXI. Y para los puntos 2, 4 y 5, se ha analizado que estas necesidades son solicitudes particulares debido a la forma de trabajo de este grupo de colaboradores. Si se implementan en ELXI, algunas de las actividades serían restringidas pero es probable que otros grupos de trabajo con diferentes necesidades las requieran. Por lo tanto, se propone que se incorpore a ELXI, como parte de un trabajo a futuro, una herramienta en la cual se configure las responsabilidades y permisos de los colaboradores, es decir, que en cada grupo se cree un diagrama de jerarquías en el que se especifique si existe una máxima autoridad y quienes serán los colaboradores subordinados. Asociado a cada elemento de este diagrama, se especificarían las actividades que pudiera desempeñar el autor en el documento. De esta forma, el sistema ELXI continuaría siendo un sistema flexible y permitiría que su uso no se limite a una sola institución, sino que se pueda implementar en cualquiera del sector público o privado.
Con lo que respecta a la prueba de desempeño, el esquema de **documento-archivo** sigue siendo una buena opción en ELXI, ya que existen diferencias de menos de 2 segundos entre el tiempo de almacenamiento del contenido de un fragmento con el esquema **fragmento-archivo**. Pero si se desea obtener el tiempo que se pierde con el esquema **documento-archivo,** entonces es conveniente implementar el **fragmento-archivo,** considerando una reestructuración en el sistema de archivos, así como en los procesos de creación, actualización, borrado en el archivo, y la construcción de la visualización.

Capítulo 5

Conclusiones

Del trabajo desarrollado en esta tesis se puede concluir que la implementación del mecanismo de control de concurrencia, ha reforzado la fase de escritura en el sistema ELXI. Esto es debido a que el mecanismo que controla el proceso de escritura asíncrona se basa en tres esquemas que se resumen de la siguiente manera:

- Estructuración del documento. En este esquema se ha definido que un documento puede estar constituido por uno o más fragmentos. Gracias ha esta estructuración es posible que el sistema detecte en que parte del documento esta escribiendo un colaborador.
- Asignación de responsabilidades a los colaboradores, mediante roles de documentos y fragmentos. Con este esquema se establece cuales son los permisos de acceso de un colaborador tanto en un documento como en los fragmentos que lo constituyen. Asimismo, se incorpora el concepto de grupo de trabajo que también controlo el acceso a los documentos. En la evaluación del mecanismo se pudo comprobar que los grupos de trabajos y la asignación de roles de documento y fragmentos controlaron efectivamente el acceso a los documentos, así como el de establecer que colaborador puede ser un escritor potencial.
- Control de escritura sobre la unidad de concurrencia. Para ello se codificó el mecanismo de bloqueo, el cual se encarga de conceder el permiso para realizar el proceso de escritura en un fragmento cuando ningún otro colaborador esta escribiendo en él. Este mecanismo también verifica que el colaborador que ha solicitado escribir tenga el rol de escritor en dicho fragmento. Con esto, se garantiza que ningún colaborador que no sea escritor en un fragmento pueda editarlo. De igual manera, se asegura que sólo un colaborador pueda escribir en un fragmento a la vez, con lo que las aportaciones de dicho colaborador no se perderán o no serán alteradas, si otro colaborador escribe en otra parte del documento al mismo documento.

De igual manera, se consideró el caso de las solicitudes de bloqueo rechazadas. Para ello, se implementó una lista de espera de tipo FIFO, en donde la primera solicitud en llegar es la primera que se atiende cuando se libera el bloqueo. Por lo tanto, aún cuando un colaborador que tenga el permiso de escritor no haya obtenido el bloqueo del fragmento porque éste ha sido bloqueado por otro escritor, tendrá la posibilidad de establecer que en un turno siguiente el escribirá en el fragmento.
Es importante recordar que los colaboradores no trabajan con los documentos originales, sino con copias. Sin embargo, cuando los colaboradores deciden guardar sus cambios permanentemente,

estos los hacen efectivos en los documentos originales. Por ello, es importante controlar el proceso de guardar el documento en cada una de las solicitudes clientes.

5.1 Desarrollo e implementación

Antes de implementar el mecanismo de control de concurrencia se consideró la reestructuración de la base de datos, la ruta de almacenamiento de los documentos en el sistema de archivos y los documentos.
El objetivo de la base de datos siguió siendo el mismo, almacenar información relacionada con los colaboradores y los documentos. Sin embargo, en la reestructuración se tuvo que eliminar la tabla ***usuario_archivo*** e incorporar cuatro más. Esto se debió a que surgieron las entidades miembros, grupos, fragmentos y roles. Así también se establecieron nuevas relaciones.
Con respecto a la ruta de almacenamiento de los documentos en el sistema de archivos, fue modificada al incorporar el concepto de grupo de trabajo. Ahora, los documentos residen dentro de un directorio llamado con el nombre del grupo al que pertenecen. De esta forma, el sistema ELXI organiza los documentos por grupo, proporcionando también una fácil ubicación cuando se quiera acceder a ellos.
Debido a que los documentos están constituidos por fragmentos, se tuvo que definir cual era el nuevo esquema de la DTD y la estructura del documento XML. Para la DTD se definieron los elementos ***documento*** y ***fragmento***, cada uno con sus atributos. Es importante recordar que la DTD es utilizada para validar los documentos XML, así que cualquier documento que tenga elementos que no estén definidos en la DTD, no podrá ser leído por el sistema ELXI.
Existe otro documento llamado *user_info*, el cual es utilizado para almacenar la información del colaborador. Este también fue reestructurado al igual que su DTD, la cual incorporó elementos relacionados con las entidades grupo, documento y fragmento. Cabe recordar que el documento *user_info* es utilizado por el proceso cliente en modo sin conexión, para saber cuales son los grupos, documentos y fragmentos a los que tiene acceso un colaborador.
Como se puedo observar el mecanismo de control de concurrencia se implementó mediante el lenguaje de programación JAVA. Esto se debe a que la arquitectura base esta construida en ése lenguaje, además de que se utilizaron algunas clases existentes, principalmente aquellas que trabajan con los documentos y la base de datos.
Para implementar el manejo de grupos y asignación de roles se incorporaron las siguientes clases al sistema ELXI:

1. **systemGroup**. Clase utilizada para crear una ventana donde se crean y eliminan grupos.
2. **groupUsers**. Clase utilizada para crear una ventana donde se incluyen o excluyen miembros de un grupo.
3. **DocumentUser**. Clase utilizada para crear una ventana donde se asignan los roles de documento.
4. **systemPermissions**. Clase utilizada para crear una ventana donde se asignan los roles de fragmento.

En el manejo de documentos se anexaron las clases:

5. **fragObj**. Clase utilizada para crear un objeto que almacena información de un fragmento.
6. **timeDialog**. Clase utilizada para crear una ventana donde se establece el tiempo de bloqueo de un fragmento.
7. **waitThread**. Clase utilizada para crear un hilo, cuya tarea es la informar al colaborador que el tiempo de bloqueo de un fragmento va acabar en los próximos 10 minutos.

Para el mecanismo de bloqueo se crearon las siguientes clases:

8. **lockedThread**. Clase utilizada para crear un hilo, cuya tarea es contabilizar el tiempo de bloqueo de un fragmento.
9. **LinkedQueue**. Clase utilizada para crear la lista de espera cuya estructura es de tipo FIFO.
10. **LinkedQueueElement**. Clase utilizada para definir las posiciones de los elementos de la lista de espera.
11. **LinkedIterator**. Clase utilizada para enlazar los elementos de la lista de espera.
12. **ObjectQueue**. Clase utilizada para crear un elemento de la lista de espera.

Además, se modificaron algunas clases existentes como:

13. **fileObj**. Clase utilizada para crear un objeto que almacena la información de un documento.
14. **DocumentUser**.. Clase utilizada para el manejo de documentos.
15. **xmlEdit**. Clase utilizada para crear la ventana del editor y visualizar el documento.
16. **elxiCliente**. Clase utilizada para crear el proceso cliente.
17. **ServidorElxiImp**. Clase utilizada para crear el proceso servidor.

Para evitar conflictos de escritura y lectura de un documento a bajo nivel, se recurrió al mecanismo de bloqueo de JAVA . Este bloqueo obedece a un protocolo de adquisición y liberación controlado por medio de la palabra clave ***synchronized*** y funciona únicamente en el contexto de los hilos.

5.2 Resultados y limitaciones

En los resultados de las pruebas de usuario se verificó el funcionamiento del mecanismo de control de concurrencia, así como, la aceptación de éste entre los colaboradores. Mediante el cuestionario que se les presentó, se observó que las siguientes medidas implementadas en el sistema ELXI tuvieron una buena aceptación:

- Organización del trabajo a través de grupos.
- Estructuración de documentos en fragmentos.
- Asignación de roles a los colaboradores sobre documentos y fragmentos.
- El proceso de bloqueo.
- Visualización del estado del fragmento.
- Reservación del bloqueo.

Sin embargo, hubo varios comentarios para complementar la aplicación y mejorarla. Éstos se resumen en los siguientes puntos:

- Mejorar el proceso de agregar miembros a un grupo.
- Incorporar el permiso de crear grupo a los colaboradores, asignado por un administrador.
- Incluir una herramienta de planeación.
- Permitir al administrador del documento cambiar las solicitudes de los colaboradores en la reservación del bloqueo.
- Permitir la operación de borrado de documento o fragmento al administrador del documento sin considerar la contribución de otros autores y la opinión del administrador del fragmento.

Con lo que respecta a la prueba de desempeño, el esquema de **documento-archivo** sigue siendo una buena opción en ELXI. La diferencia que tiene con respecto al esquema **fragmento-archivo** es que el tiempo de almacenamiento fue un poco menor (menos de 2 seg. en una máquina pentium III a 400 Mhz), considerando que el número máximo de colaboradores que trabajan al mismo tiempo en un documento es de 30. Pero si se desea recuperar el tiempo que se pierde con el esquema **documento-archivo,** entonces es conveniente implementar el **fragmento-archivo,** considerando una reestructuración en el sistema de archivos, así como en los procesos de creación, actualización, borrado en el archivo y la construcción de la visualización.

5.3 Trabajo a futuro

Haciendo referencia a los resultados obtenidos en la etapa de evaluación, se esta trabajando en una nueva versión en ELXI, en la cual se esta considerando mejorar el proceso de agregar miembros a un grupo, así como el de incluir una herramienta de planeación. Por el momento, los colaboradores tendrán que reunirse frente a frente o utilizar otra aplicación para decidir quien será el encargado de crear el grupo de trabajo, administrar el documento y cuales serán los roles que se les asignarán a los colaboradores.
Así también, se esta considerando la integración de una herramienta en la cual se configuran las responsabilidades y permisos de los colaboradores, es decir, que en cada grupo se cree un diagrama de jerarquías en el que se especifique si existe una máxima autoridad y quienes serán los colaboradores subordinados. Asociado a cada elemento de este diagrama, se especificarían también las actividades que pudiera desempeñar el colaborador en el documento. De esta forma, el sistema ELXI continuaría siendo un sistema flexible y permitiría que su uso no se limite a una sola institución, sino que se pueda implementar en cualquiera del sector público o privado.

5.4 Contribución de tesis

En este trabajo de tesis se logró un desarrolló teórico y práctico que contribuye al campo de investigación del CSWriting. En el desarrollo teórico se pudo constatar que el diseño de los sistemas groupware no sólo se basan en investigaciones realizadas en el área del CSCW o HCI, sino también se requieren otros fundamentos que resuelvan problemas de heterogeneidad,

seguridad, escalabilidad, fallas, concurrencia y transparencia, los cuales son temas de estudio de los sistemas distribuidos. Asimismo, se hizo referencia a un tipo de sistemas en particular que forman parte del campo de estudio de la disciplina *CSCWriting*, los sistemas de edición colaborativa. En ellos se señaló que en su arquitectura pueden existir diferentes mecanismos implementados para el control de documentos. Sobre todo, en aquellos sistemas que apoyan la edición concurrente. Algunos editores proponen esquemas basados en responsabilidades, mediante roles o permisos, o bien, mecanismos de control de concurrencia (por ejemplo bloqueo de escritura sobre fragmentos, bloqueo sobre texto seleccionado, transformación de operaciones, etc.) Sin embargo, existe la posibilidad de que no sea suficiente implementar uno de estos mecanismos.

En el desarrollo práctico existe una contribución con respecto a la manera en que se realizó el análisis para proponer una solución a los problemas que surgen en la edición concurrente de documentos en el sistema de autoría colaborativa ELXI. Así también, se explica como se puede implementar la solución utilizando la metodología orientada a objetos y el lenguaje UML, mediante los cuales se describe de manera ilustrativa los procesos que ejecutan el mecanismo de control de concurrencia. De igual manera, quedan descubiertas algunas estrategias de programación.

Referencias bibliográficas

Libros y articulos

[App87] Apple Computer Inc. **Human Interface Guidlines: The Apple desktop Interface.** Adison Wesley, 1987, pp. 144.

[BePo93] Ronald M. Baecker, Dimitrios Nastos, Ilona R. Posner y Kelly L. Mawby. **The User-Centred Iterative Design of Collaborative Writing Software**. INTERCHI'93. 1993, pp. 399-405.

[Bac92] Bacon Jean, Concurrente Systems. **An Aproach to Operating Systems, Database and Distributes Systems**. Addison Wesley, 1992, pp.156,157.

[Bae94] Ron Baecker, Geof Glass, Alex Mitchell y Ilona Posner. **SASSE: The Collaborative Editor. ACM, Conference Companion**, CHI'94, E.U.A., abril 1994.

[Bai85] J. H. Bair. **The Need for Collaboration Tolls in Offices**. En Proceedings of AFIPS'85 Office Automation Conference, Georgia, E.U.A. 1985, pp. 59-68.

[Bec93] E. Beck. **A Survey of Experiences of Collaborative Writing**. Computer Supportes Collaborative Writing, editor: M. Sharples, Springer-Verlag,1993, pp.87-112.

[BeBe93] M. Beck y V. Belloti. **Informed Opportunism as Strategy: Supported Coordination in Distributed Collaborative Writing**. ECSCW'93 Proceedings of the third European Conference on Computer Supported Cooperative Work, Kluwer Academic Publishers, Londres, Inglaterrra, 1993, pp. 233-248.

[Bri87] Bridwell-Bowles, L. Johnson, y Brehe. **Composing and Computers: Case Studies of experieced Writers**. Writing in Real Time: Modelling Production Processes.1987, pp. 81-107.

[CePe85] Ceri, S. y Pelagatti, G. **Distributed Databases - Principles and Systems**. MacGraw-Hill, 1985.

[Cha92] Chandler D. **The Phenomenology of Writing by Hand**. Intelligent Tutoring Media. vol. 3(2/3), 1992, pp.65-74.

[Cou01] George Coulouris. **Distributed System. Concepts and Design**. Addison Wesley, 3ª edición, 2001.

[Dei93] Deitel H. **Sistemas operativos**. Addison Wesley, 2ª edición, 1993.

[DoBe92] Paul Dourish y Victoria Bellotti. **Avareness and Coordination in Shares Workspaces.** Proceedings of the CSCW'92, Toronto Canadá, 1992, pp. 107-114.

[EdLu90] L. Ede y A. Lunsford. **Singular Text/Plural Authors: Perpectives on Collaborative Writing**. Southern Illinois University Press. Illionois, E.U.A. 1990.

[Ell91] Clarence Ellis, Simon Gibs y Gail Rein. **Groupware, Some Issues and Experiences**. Communications of the ACM, vol. 34, No. 1, enero 1991, pp. 30-37.

[Gre97] Greenberg, S. **Collaborative Interfaces for the Web**. In C. Forsythe, E. Grose and J. Ratner (editors), Human Factors and Web Development, Chapter 18, 1997, pp. 241-254.

[Gut96] Carl Gutwin, Saul Green y Mark Roseman. **Workspace Awareness in Real-Time Distributed Groupware: Framework, Widgets, and Evaluation.** Proceedings of the HCI'96, Londres U.K, pp. 281-298.

[HaFl80] J.R. Hayes y L. S. Flower. **Identifying the Organization of Writing Process**. En Cognitive Process of Writing, New Jersey, E.U.A. 1980, pp.3-30.

[HaRe83] Härder, T. y Reuter, A. **Principles of Transaction-Oriented Database Recovery**. Computing Surveys, Vol.15, No. 14, 1983.

[HaRe83] Henry Pérez Luna. **Un Espacio de Trabajo Compartido como Apoyo a la Edición Colaborativa en Internet**. Tesis de Maestría, Posgrado en Ciencia e Ingeniería de la Computación, México, 2001.

[JaLo92] Jarczyk A., Löffler P., Völksen G. **Computer Supported Cooperative Work (CSCW) State of the Art**. Version 1.0, Internal Report, Siemens AG, Munich, 1992.

[LiMu98] Du Li and Richard R. Muntz. COCA: **Collaborative Objects Coordination Architecture.** In Proceedings of ACM CSCW '98 Conference on Computer Supported Cooperative Work, Seattle, Washington, November 1998.

[Kra90] R. E. Kraut, C. Egido y J. Galegher. Patterns of Contact and Communication in Organizations: Form, function and technology. En S. Oskamp y S. Spacapan, People's Reactions to Technology, Newbury Park: Sage Publications, 1990, pp.145-149.

[KuRo81] Kung, H. T. y Robinson, J.T. **Optimistic methods for concurrency control. ACM Transactions on Database Systems**. vol.6, No. 2, 1981, pp.213-226.

[Mil94] Milenkovic M. **Sistemas Operativos. Conceptos y Diseño**. 2ª Edición, McGraw-Hill, 1994.

[Mit95] Alex Mitchell, Ilona Posner y Ronald Baecker. **Learning to Write Together Using Groupware**. Conference proceedings on Human factors in computing systems, 1995, pp. 288 -295.

[Mat81] Matsuhashi A. **Pausing and Planning: The Tempo of Written Discourse**. Research in the Teaching of English. vol. 15(2), pp. 113-134.

[MuCe] César Murillo. **Diseño e implementación de mecanismos de consistencia en un sistema de edición colaborativa en Internet**. Tesis de maestría, Instituto de Investigaciones de Matemáticas Aplicadas y Sistema, México.

[NePe91] R. E. Newman-Wolfe y Harsha K. Pelimuhandiram. **MACE: a fine grained concurrent editor**. Conference proceedings on Organizational computing systems. 1991, pp. 240-254

[NeRa91] Newman-Wolfe, C. Ramírez, H. Pelimuhandiram, M. Montes, M. Webb y D. Wilson. **A brief Overview of the DCS Distributed Conferencing System**. Proceedings of Summer Usenix Conference, Nashville,TN, june 1991, pp 437-452.

[Neu94] Christine Neuwirth, David Kaufer, Ravinder Chandhok y James Morris. **Computer Support for Distribuited Collaborative Writing: Defining Prarameters of Interaction**. Proceedings of the CSCW'94, Chapel Hill USA, pp. 145-152.

[Nys89] M. Nystrand. **A Social-Interactive Model of Writing**. En Written Communication, vol. 6, 1989, pp. 66-85.

[PaSa94] François Pacull, Alain Sandoz y André Shiper. **Duplex: A Distributed Collaborative Editing Environment in Large.** Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW), 1994.

[PoBa92] Ilona Postner y Ronald Baecker. **How People Write Together**. Proceedings of the 25th Hawaii International Conference on system Sciences, vol. 4, enero de 1992, pp. 127-138.

[Por99] Roberto Portugal, Luis Guerrero, David Fuller. **A Customizable Collaborative Virtual Enviroment on the Web.** Proceedings of the CRIWG'99, Cancún México, pp. 328-335.

[ReSe94] Walter Reinhard, Jean Schweitzer y Gerd Völksen. **CSCW Tools: Concepts and Arquitectures.** IEEE COMPUTER, May 1994.

[Rom97] Manuel Romero Salcedo y Dominique Decouchant. **Structures Cooperative Authoring for the World Wide Web.** Computer Supported Cooperative Work:The Journal of Collaborative computing, 1997, pp.157-174.

[Rom98] Manuel Romero. **Alliance sur L'Internet: support pour lédition coopérative de documents structures sur un réseau a grande distance**. Tesis de Doctorado, Instituto Nacional Politécnico de Grenoble, Francia, abril de 1998.

[RoRa03] M. Romero-Salcedo, S. Ramírez, C. Murillo and A. Avelar, "**ELXI: un Editor CoLaborativo de documentos Xml en Internet para la elaboración de documentación técnica en el IMP**," Intelligent Computing for the Petroleum Industry. Ed. Instituto Mexicano del Petróleo, noviembre del 2003, pp. 25-26.

[RoAn] Aníbal Avelar Rosales. **Diseño, implementación y evaluación de un editor de documentos XML en Java para el sistema de autoría colaborativa ELXI.** Tesis de maestría, Instituto de Investigaciones de Matemáticas Aplicadas y Sistema, México.

[RuFa98] Diana Ruiz y Jesús Favela. **Collaborative Review and Edition of HTML Documents.** Proceedings of the 4th International Workshop on Groupware, Buzios Brasil, 1998, pp. 113-128.

[Sch96] Christian Schuckmann, Lutz Kirchner, Jan Schümmer, Jörg M. Haake. **Designing object-oriented synchronous groupware with COAST.** Integrated Publication and Information Systems Institute, 1996.

[Sha93] Mike Sharples, J. Goodlet, E. Beck, C. Wood, S. Easterbrook y L. Plowman. **Research Issues in the Study of computer Supported Collaborative Writing.** Computer Supported Collaborative Writing, Springer-Verlag Alemania, 1993, pp. 9-28.

[Sha94] Mike Sharples. **Computer Support for the Rhythms of Writing**. Computers and Composition, vol.11, 1994, pp. 217-226.

[Sik97] Klass Sikkel. **A Group-based Authorization Model for Cooperative Systems**. Proceedings European Conference on Computer-Supported Cooperative Work (ECSCW'97), Lancaster, 1997.

[Stal95] Stallings, W. **Computer Organization and Architecture**. 3ª Edición, New York: Macmillan, 1993.

[Ste87] M. Stefik, D. G. Bobrow, G. Foster, S. Lanning y D. Tratar. **WYSIWIS Revised: Early Experiences with Multiuser Interfaces**. ACM Transactions on Information Systems, 1987, pp. 147-167.

[SuBu98] Tamara Sumner, Simon Buckingham Shum. **From Documents to Discourse: Shifting Conceptions of Scholarly Publishing**. Human Factors in Computing Systens(CHI'98), Abril 1998.

[Sus99] David Sussman. **WebDAV: A Panacea for Collaborative Authoring?** IEEE MultiMedia, 1999, pp. 76-79.

[Tan98] Tanenbaum, A. **Sistemas Operativos**. Tercera Edición. Prentice Hall.1998.

[Tre94] Jonathan Trevor. **Infrastructure Support for CSCW**. Submitted for the degree of Doctor of Philosophy. Department of Computing, Lancaster University, U.K. December 1994.

[TrRo93] J Trevor, T Rodden, G Blair. **COLA: A lightweight platform for CSCW**. In: Proc of European Conf on CSCW, Milan, 1993, pp.15-30

[Vol93] Völksen, G. **Approach Strategies to Groupware**. Proccedings Groupware '93 Europe Conferences in Stockholm, London, Frankfurt, The Conference Group, Scottsdale, AZ, 1993.

Direcciones URL

[WBSC] http://bscw.gmd.de/

[WCOR] http://www.corba.org

[WDco] http://www.microsoft.com/com/tech/DCOM.asp

[WDou] http://www.ibiblio.org/pioneers/englebart.html

[WGrK] http://www.cpsc.ucalgary.ca/grouplab/projects/GroupKit.html

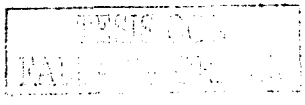
[WGro] http://www.groove.net

[WHyn] http://www.hypernews.org/HyperNews/get/hypernews.html

[WJav] http://java.sun.com/products/jdk/rmi/examples.html

[WMus] http://www.dcs.qmul.ac.uk/research/distrib/Mushroom/

[WNet] http://java.sun.com/products/jdk/rmi/examples.html



[WNHa] http://havefun.ncsa.uiuc.edu/habanero/

[WNov] http://www.novell.com/news/press/pressroom/history.html

[WTeW] http:// www.teamwave.com

[WXde] http://www.xcf.berkeley.edu/~jmacd/xdelta.html