Trabajo de Grado

"Herramientas para Ambientes Colaborativos"

Autoras:
Claudia Graciela de Sousa Rodrigues
Claudia Mariel Roldán

Director
Luis Mariano Bibbó

Co-Director
Gustavo Rossi

Facultad de Ciencias Exactas
Universidad Nacional de La Plata

1997

Indice

Introducción v

Agradecimientos vi

Capítulo 1: Apreciaciones Generales

1.1.- Introducción 1
1.2.- Herramientas para colaborar 2
1.3.- Sociología de la colaboración 3
1.4.- Primero colaboración, segundo computación 3
1.5.- Taxonomía para ambientes colaborativos 5
1.6.- Requerimientos para aplicaciones cooperativas 6

Capítulo 2: Distintas Aplicaciones de la Colaboración

2.1.- Colaboración en la educación 8
2.2.- Colaboración en la empresa 10

Capítulo 3: Toma de decisiones en grupo

3.1.- El trabajo en grupo 13
3.2.- Toma de decisiones 13
3.2.1.- Modelos 14
3.2.1.1.- Modelo racional 14
3.2.1.2.- Modelo político 15
3.2.1.3.- Modelo de proceso 15
3.2.1.4.- Modelo tarro de residuos 16
3.2.2.- Fases 16
3.2.2.1.- Fase de entrada 17
3.2.2.2.- Fase de procesamiento 17
3.2.2.3.- Fase de salida 18
3.2.2.4.- Fase de revisión 18
3.2.3.- Estrategias para la toma de decisiones 18

Capítulo 4: Internet

4.1.- Introducción 20
4.2.- Historia 20
4.3.- ¿Qué la compone? 22
4.4.- ¿Quién la gobierna? 22
4.5.- ¿Quién paga? 23
4.6.- ¿Qué significa para el usuario? 23
4.7.- ¿Qué se trae el futuro? 24

Capítulo 5: Groupware

5.1.- Introducción 26
5.2.- Definición 27
5.3.- Esquemas de clasificación 28
5.4.- El mercado 29
5.5.- Tendencias 29
5.6.- Motivaciones 30
5.7.- Desafíos 31
5.8.- Comunicación, Colaboración y Coordinación 31
5.8.1.- Comunicación 32
5.8.1.1.- Definición 32
5.8.1.2.- Envío de mensajes electrónicamente: una tecnología para la comunicación 32
5.8.1.3.- Almacenamiento de mensajes 32
5.8.1.4.- Conclusiones 34
5.8.2.- Colaboración 34
5.8.2.1.- Introducción 34
5.8.2.2.- Bases de datos compartidas: una tecnología para la colaboración 34
5.8.2.3.- Aplicaciones colaborativas 35
5.8.2.4.- Bases de datos compartidas en otras aplicaciones 36
5.8.2.5.- Sistemas de publicación de referencias 36
5.8.2.6.- La naturaleza pasiva de las bases de datos compartidas 37
5.8.2.7.- WWW como una herramienta de colaboración 37
5.8.2.8.- Conclusiones 39
5.8.3.- Coordinación 40
5.8.3.1.- Introducción 40
5.8.3.2.- Cuatro generaciones de envío de mensajes 41
5.8.3.3.- Integración de mensajes y groupware 41
5.8.3.4.- Modelo de mensajes del flujo de trabajos 41
5.8.3.5.- Modelo de bases de datos compartidas para flujo de trabajos 42
5.8.3.6.- Un modelo integrado 42
5.8.3.7.- Modelo de transacciones extendido 43
5.8.3.8.- Entorno para el desarrollo de aplicaciones 43

Capítulo 6: Trabajo Colaborativo Soportado por Computadoras

6.1.- Definición 45
6.2.- Un poco de historia 45
6.3.- Contextos de investigación y desarrollo: historia y contribuciones 46
6.4.- Cooperación, coordinación y control en el trabajo con CSCW 47
6.5.- Contradicciones de las revoluciones tecnológicas y las transformaciones sociales 49

Capítulo 7: AMCO: Una Herramienta para Colaborar - Visión Externa

7.1.- Introducción 50
7.2.- Generalidades 50

7.3.- Especificación 50
7.3.1.- Manejador de Páginas 51
7.3.2.- Chat 52
7.3.3.- Visor 52
7.4.- Colaboración 52
7.5.- Visualización y Explicación 54
7.5.1.- Pantalla Inicial 54
7.5.2.- Pantalla Principal 56
7.5.2.1.- Ventana Visor 57
7.5.2.2.- Ventana Chat 58
7.5.2.3.- Ventana Manejador de Páginas 59
7.5.2.3.1.- Ventana de Inicio 59
7.5.2.3.1.1.- Ventana de Inscripción 60
7.5.2.3.2.- Ventana Principal 62
7.5.2.3.2.1.- Ventana Proponer Temas 62
7.5.2.3.2.2.- Ventana Temas Propuestos 63
7.5.2.3.2.3.- Ventana Votar Aspirante 64
7.5.2.3.2.4.- Ventana Crear Páginas 65
7.5.2.3.2.5.- Ventana Votar Páginas 67
7.5.2.3.2.6.- Ventana ¿Soy Autor? 68
7.5.2.3.2.7.- Ventana Modificar Datos 69

Capítulo 8: AMCO: Una Herramienta para Colaborar - Visión Interna

Sección I

8.1.- Objetos Distribuidos 71
8.1.1.- Ideas Generales 71
8.1.2.- Ventajas técnicas de los objetos distribuidos 72
8.2.- Java 73
8.2.1.- Simple y Familiar 73
8.2.2.- Orientado a Objetos 74
8.2.3.- Arquitectura Neutral 77
8.2.4.- Portable 78
8.2.5.- Robusto 78
8.2.6.- Interpretado y Dinámico 78
8.2.7.- Seguridad 79
8.2.8.- Multithreading 80
8.2.9.- Performance 81
8.3.-RMI (Remote Method Invocation) 81
8.3.1.- Introducción 81
8.3.2.- Objetivos del Sistema 82
8.3.3.- Modelo de Objetos Distribuido Java 82
8.3.3.1.- Contraste entre los modelos distribuidos y no distribuidos 82
8.3.3.2.- Clases e interfaces RMI 83
8.3.3.2.1.- La interface Remote 84
8.3.3.2.2.- La clase RemoteException 84
8.3.3.2.3.- La clase RemoteObject y sus subclases 84

8.3.3.3.- Implementación de una interface remota 85
8.3.3.4.- Equivalencia de tipos de objetos remotos con stubs locales 85
8.3.3.5.- Pasaje de parámetros en RMI 86
8.3.3.5.1.- Pasaje de objetos no remotos 86
8.3.3.5.2.- Pasaje de objetos remotos 86
8.3.3.6.- Manejo de excepciones en RMI 86
8.3.3.7.- Sobreescritura de métodos por RMI 86
8.3.3.8.- Semántica de métodos de Object declarados como final 87
8.3.3.9.- Localización de objetos remotos 87

Sección II

8.4.- AMCO: Visión Interna 87
8.4.1.- Requerimientos 87
8.4.2.- Lenguaje de implementación 88
8.4.3.- Arquitectura 88
8.4.4.- Diseño 90

Conclusiones 92

Apéndice A: Java 94

Apéndice B: Invocación de Métodos Remotos (RMI) 112

Apéndice C: Casos de Uso y Diseño Orientado a Objetos 131

Bibliografía 139

Introducción

Nuestra tesis se basa en estudiar la colaboración entre personas, la cual ocurre a partir de la interacción directa entre miembros de un equipo. La colaboración ayuda a solucionar problemas, ya que facilita el éxito en la performance y permite incrementar el aprendizaje, gracias al conocimiento obtenido de la experiencia de otros integrantes del grupo. Dichos grupos pueden solucionar problemas más interesantes y complejos que el trabajo individual aislado. Las personas que trabajan en grupos necesitan articular diseños, críticas y argumentos para comunicárselos a otros miembros, dando impulso a la clase de reflexión que lleva al aprendizaje.

La tendencia hacia la colaboración es muy fuerte, empujada desde el lado de la tecnología con la proliferación de las redes. Esto, unido a la multimedia, la programación visual y la programación orientada a objetos, facilita la integración entre los individuos. Además, surge Internet como un medio de comunicación muy fuerte, debido a que cada vez más personas en el mundo se conectan a ella. Así, se abre una mayor posibilidad de crear aplicaciones colaborativas permitiendo que, personas que se encuentran en lugares geográficos diferentes, puedan intercambiar opiniones, compartir trabajos, generar resultados cooperativos,.... De esta manera, se logra una nueva forma de acercamiento entre personas, pudiéndose solucionar problemas destinados a ser tratados por un solo individuo, en forma mucho más rápida y eficiente, por haber muchas personas trabajando en él.

El desarrollo de aplicaciones colaborativas implica el uso de modelos distribuidos. Además, es conveniente la utilización de un modelo orientado a objetos pues mediante esta tecnología se puede reflejar más naturalmente el mundo real. Por lo tanto, como la aplicación es distribuida y se basa en un modelo orientado a objetos, es obvia la necesidad de usar un mecanismo de objetos distribuidos, el cual trae aparejado los beneficios de standarización, bajos costos de mantenimiento, alta reusabilidad y gran independencia de la plataforma. Por otra parte, se hace transparente al programador la complejidad de los mecanismos de comunicación subyacentes, pues interactúa con objetos remotos vía invocaciones a métodos familiares, de la misma manera que si fueran locales.

Hemos desarrollado una aplicación, a modo de demostración, sobre algo de lo que se puede hacer en favor de la colaboración, aprovechando la comunicación brindada por Internet. Elegimos JAVA como lenguaje de programación porque es simple, familiar, orientado a objetos, de arquitectura neutral, portable, robusto, interpretado, dinámico, seguro y de alta performance. Nos enfocamos principalmente en su paquete RMI, el que permite hacer invocaciones a métodos remotos. La aplicación consta de un ambiente que reúne tres componentes:

- un chat, el que permite que varias personas que se encuentran en lugares geográficos diferentes puedan intercambiar mensajes,
- una herramienta para tomar decisiones en grupo, usando como tema base el manejo de páginas HTML.
- un visor para poder observar el estado en que se encuentra cada usuario del ambiente.

Agradecimientos

Queremos dar gracias a los integrantes del L.I.F.I.A por habernos permitido contar con toda su infraestructura, en particular, a los integrantes del L.I.F.I.A Multimedia, quienes nos brindaron el lugar para trabajar en nuestra tesis.

Capítulo 1

Apreciaciones Generales

Colaborar es trabajar con otra u otras personas en una misma tarea .....

1.1.- Introducción

La colaboración tiene lugar a través del tiempo, construyendo conocimiento basado en el trabajo anterior, y ocurre a partir de la interacción directa entre miembros de un equipo.

La colaboración se basa en un espacio compartido, que puede ser una habitación, un pizarrón, un espacio compartido on-line, etc. Dicho espacio sirve como puntapie inicial del acto de colaborar y es esencial como medio para manejar la ambigüedad propia de la interacción humana. [1]

La colaboración se puede dar entre dos personas, o bien tratar de compartir información de muchos a muchos. [1] Si una única persona tuviera toda la capacidad intelectual y el tiempo necesario para desarrollar un proyecto, probablemente obtendría máxima coherencia e integridad. Sin embargo, las restricciones del mundo real a menudo impiden esta posibilidad. Ningún individuo tiene toda la experiencia, información, diversidad de puntos de vista, o el tiempo para llevar a cabo tales proyectos. En su lugar, estos recursos deben sintetizarse en grupos, los que estarían formados por miembros que tienen diferentes clases de conocimiento y destreza requeridos por el proyecto, y cuyo esfuerzo colectivo podría completar la tarea en un período de tiempo más corto. Sin embargo, cuando los grupos comienzan a crecer en cantidad de participantes, el tiempo en realizar la tarea en sí misma decrece, mientras que el tiempo de comunicación entre los miembros del grupo aumenta. El problema se agrava, si los individuos que trabajan juntos se encuentran geográficamente diseminados. Estas separaciones físicas, aunque sean relativamente pequeñas, pueden inhibir seriamente las interacciones entre las personas. Para resolver estos problemas, se necesitan nuevas herramientas que aumenten el proceso de colaboración. [3]

Una cuestión fundamental es, cómo colaboran los grupos para construír grandes estructuras conceptuales coherentes. En este punto, se debe distinguir entre colaboración y cooperación:

- La colaboración se basa en un único propósito y en la integración de las partes, sin fisuras, como si el objeto conceptual fuera producido por una única mente. Por ejemplo, un documento colaborativo tendrá un claro mensaje o propósito. El lector no será capaz de decir cuáles de los capítulos fueron escritos por algunos de los posibles autores y cuáles por otros. Las secciones serán también consistentes unas con otras, y una sección mostrará conocimiento apropiado del contenido de otras secciones.

- La cooperación es menos restrictiva en la demanda de integración intelectual. Requiere de un conjunto de grupos o de un conjunto de individuos que comprendan un único grupo y que lleven a cabo sus tareas individuales en forma coordinada de acuerdo con algún plan. Sin embargo, en una estructura cooperativa, los diferentes individuos o grupos no necesitan conocer qué se está desarrollando en otras partes del proyecto, siempre que ellos lleven a cabo satisfactoriamente su propia tarea.

La tarea central que enfrentan los colaboradores es la de unir sus experiencias individuales para, primero, construír una estructura conceptual coherente, y luego, expresarla en palabras, dibujos, programas, etc. Los miembros del equipo - particularmente cuando provienen de diferentes disciplinas o áreas de competencia- deben construír un entendimiento común de los conceptos y términos claves, trasladando la jerga no familiar de otras disciplinas, en términos y conceptos entendibles por ellos. A través de diálogos, el grupo deberá alcanzar un consenso dentro de un ambiente de trabajo común, al cual los individuos harán sus contribuciones y del cual tengan, probablemente, un entendimiento parcial. La reunión del total de las estructuras conceptuales individuales y compartidas, evolucionará si la colaboración entre los miembros aumenta. La clave para estudiar este proceso es proyectar la relación del desarrollo entre una estructura común de ideas compartidas y las distintas subestructuras de ideas y conocimientos poseídos por los individuos del grupo. [3]

1.2.- Herramientas para colaborar

En un ambiente colaborativo, las personas interactúan entre sí, intercambiando ideas y conocimientos. Estas interacciones tienen lugar utilizando alguna herramienta de comunicación como, encuentros cara a cara ocasionales, y cambios más frecuentes por teléfono, fax y correo electrónico. Las interacciones son caras y consumen tiempo. Las conversaciones telefónicas están limitadas a un modo de comunicación simple, las cuales son inefectivas para cambios visuales y otras formas de información. El correo electrónico y los fax no permiten interacciones en tiempo real. Recientemente, se introdujeron productos de videoconferencia, que solucionan algunos de estos problemas. [4] Por otra parte, Internet fue un auge como fuente de información en sus primeros años, y ahora, con la llegada del proyecto World Wide Web (WWW) y la introducción de browsers gráficos integrando el acceso a varios recursos de Internet, hay un enorme interés de explorar nuevas formas de interacción a través de la red.

Una herramienta de comunicación se transforma en una herramienta de colaboración coordinando estrategias, formas en las cuales las personas organizan su comportamiento para cumplir una tarea determinada. Las posibles estrategias de coordinación son: brainstorming, delegación de responsabilidades, centralización de la discusión en una idea, revisión, resolución de conflictos, etc. Hasta que estas características no estén incluídas en el diseño de una herramienta, ésta seguirá siendo una herramienta de comunicación y no de colaboración. [5]

Por consiguiente, es difícil proveer soporte de computación para colaboración, pues requiere un gran entendimiento de la forma en que funcionan los grupos y las organizaciones. Sin embargo, las personas, cotidianamente, interactúan con otras sin demasiado esfuerzo, produciéndose así una paradoja que será resuelta cuando se

entienda que muchas de las cosas que hacen los individuos sin esfuerzo, para las computadoras son difíciles de realizar. Se requerirá una expansión del entendimiento, para diseñar sistemas que soporten colaboración y para predecir el impacto de las tecnologías sobre los grupos y organizaciones. Por esta razón, es fundamental la investigación del comportamiento individual y grupal, y el estudio de la naturaleza de los lugares de trabajo y de las organizaciones. [7]

1.3.- Sociología de la colaboración

Las experiencias en comunicaciones vía medios electrónicos de la última década, han revelado la omnipresencia y utilidad de los controles sociales humanos, que van más allá de los contactos cara a cara. Cuando un medio técnico de discurso elimina estos controles, su rol se transforma en aparente. A este respecto, el primer lenguaje es el de los gestos y cada encuentro social tiene lugar en el contexto de este lenguaje silencioso dado por los movimientos y orientación del cuerpo, la posición espacial, la distancia en el diálogo; es decir, el contacto visual envía mensajes silenciosos que son tan significativos como la palabra. Estas son las cosas que mantienen los controles sociales en los diálogos y que aún están ausentes en los medios electrónicos.

En términos de la palabra escrita, el correo electrónico es el mayor nivelador de comunicaciones. Pero también omite las obligaciones de salutaciones sociales y el ritual del cierre de una correspondencia. Por lo tanto, los controles sociales silenciosos están ausentes en el correo electrónico. Recientemente, han aparecido en escena "sonrisas o emociones" en forma de un conjunto de caracteres especiales que pueden leerse como la representación de una cara; es la manera de expresar sentimientos en un medio que no los tiene.

Aún cuando pueda enviarse el sonido y el video de una persona a través de las comunicaciones, las limitaciones técnicas pueden hacer que este intercambio sea mucho menos que satisfactorio. La ubicación inapropiada de la cámara puede hacer parecer que el expositor siempre mira para otro lado, distinto de su auditorio. El ancho de banda limitado en la transmisión de imágenes puede producir el congelamiento de un cuadro que justo estaba capturando un parpadeo, un estornudo o un bostezo. El tamaño pequeño y la ubicación de los monitores puede promover la sensación de cabezas que hablan, lo cual disminuye al interlocutor y al mensaje. [6]

1.4.- Primero colaboración, segundo computación

La construcción de aplicaciones que impacten directamente a las personas, es muy diferente de la construcción de productos de computación de la clase de los que se estudian y enseñan en los cursos standard de computación. En la ciencia de la computación tradicional, uno se preocupa por las estructuras de datos y los códigos, la forma de asegurar que los programas hagan lo que se prometió que harían, las formas de probar que ese código funciona, las formas de sincronizar procedimientos dentro de streams de computación y datos complejos. Todo esto es importante y esencial, pero es poco comparado con todo lo que concierne a los usuarios de los programas.

Los sistemas de computación tratan de estar dirigidos a las personas, especialmente a grupos de ellas, y deben amoldarse a sus necesidades. No hay forma de que un sistema pueda trabajar bien con personas, especialmente con grupos colaborativos, sin un entendimiento total y profundo de las personas y los grupos.

Demasiado seguido, la tecnología se construye por sus propios méritos, independientemente de sus usos y de sus usuarios. Este trabajo debe ser más fino cuando la tecnología se usa para optimizar alguna computación compleja o para descubrir algoritmos o estructuras de datos. No es apropiado cuando el fundamento de la tecnología es actuar como guía directa de las actividades laborales de las personas.

Cada vez más las computadoras están haciendo diferencias entre el estilo de vida y de trabajo de las personas. Cuando esto ocurre, es hora de empezar a examinar las características que las rodean. Se deben reconocer ciertos principios como:

- No es posible entender a las personas sin una observación sistemática, ya sea en el campo, en la oficina, en la casa o en el laboratorio.

- Todos pensamos que entendemos a las personas, porque nosotros somos personas y llevamos toda la vida observándonos a nosotros mismos y a nuestros colegas. Pero no es así, pues en la psicología y en la sociología hay cosas que no entendemos o de las que tenemos creencias erróneas.

- Los mismos principios que se aplican al desarrollo de sistemas de computación para individuos solos, no son suficientes para grupos, porque las actividades de éstos son muy diferentes a las actividades individuales, y tienen sus propias necesidades y requerimientos.

- No hay manuales o resultados standard sobre los que nos podemos basar. Las ciencias sociales y cognitivas han hecho adelantos rápidos en el entendimiento, pero la distancia dejada atrás es enorme. En el presente, cuanto más básica es la función, mejor es el entendimiento. Sabemos sobre la percepción, sobre la memoria a corto plazo y sobre el entendimiento y el aprendizaje simples. Cuanto más rutinaria es la tarea a realizarse, mejor podemos definir, construír y analizar los requerimientos computacionales y de interfaz. La ciencia cognitiva no entiende el rol del stress y de las emociones. Nosotros entendemos cómo analizar palabras del lenguaje, pero no las partes no verbales que hacen mucho al mensaje, por ejemplo, los discursos, los gestos, el discurso no verbal.

- La falta de entendimiento no es excusa para evitar el tema cuando construímos sistemas de computación, ignorar estas características implica que el sistema falle.

- Los artefactos no mejoran las habilidades humanas, sino que cambian las tareas. Los artefactos basados en computadoras no son la excepción.

La tecnología sola no puede proveer las respuestas, cuando se trata de actividades humanas. Las tareas, la cultura, la estructura social, y los individuos son

componentes esenciales del trabajo, y a menos que las herramientas de computación encajen perfectamente en esta estructura, el resultado fallará.

El diseño de sistemas para trabajos cooperativos, requiere equipos de diseño cooperativo, que consistan de científicos de la computación, científicos sociales y cognitivos, y representantes de la comunidad de los usuarios. [8]

1.5.- Taxonomía para ambientes colaborativos

La taxonomía hecha por Mashayekhi para los ambientes colaborativos es:

- *En un mismo momento, en un mismo lugar*: Los participantes se juntan en una única reunión e interactúan simultáneamente. Son las reuniones convencionales.

- *En un mismo momento, en diferentes lugares*: Los participantes que están en distintas oficinas interactúan simultáneamente, llevando a cabo reuniones electrónicas. Este tipo de interacción también se denomina *colaboración sincrónica*.

- *En diferentes momentos, en el mismo lugar*: Los participantes van a una única reunión, a tiempos diferentes.

- *En diferentes momentos, en diferentes lugares*: Los participantes trabajan en sus oficinas, en el momento que ellos deciden, trabajando en forma semi-independiente. A esta división se la llama *colaboración asincrónica*.

La figura 1 expone gráficamente la taxonomía de los tipos de colaboración.

Taxonomía en tiempo y espacio

	En un mismo tiempo	**Diferentes momentos**
En un mismo lugar	Interacción cara a cara	Interacción Asincrónica
En diferentes lugares	Interacción Sincrónica Distribuida	Interacción Asincrónica Distribuida

De esta taxonomía, detallamos la segunda y la cuarta, que son las realizables por computadora.

- *Colaboración sincrónica*: Se refiere a las interacciones directas entre los miembros de un grupo, a fin de explorar, cuestionar, proponer, reveer, negociar y acordar distintos puntos dentro del trabajo común. Las

colaboraciones sincrónicas proveen la transferencia de información específica dentro del contexto de una colaboración asincrónica a largo plazo. [3]

- *Colaboración asincrónica*: Es la forma de proveer un acceso efectivo a ideas compartidas, que están siendo desarrolladas por un grupo cuyos participantes pueden variar en el tiempo, y pueden trabajar semi-independientemente y separados geográficamente.

1.6.- Requerimientos para aplicaciones cooperativas

Hay dos grupos de requerimientos para permitir configurar aplicaciones cooperativas y para construír sistemas distribuidos eficientes, seguros y amigables:

• ***Requerimientos para configurar aplicaciones cooperativas***: Son esenciales tres perspectivas para soportar cualquier clase de equipo cooperativo:

1.- ***Dinámica del grupo***:

- *Comunicación*: Tener distintos protocolos de comunicación, ofrecer varias formas de comunicación, usar medios de transmisión de mensajes tales como texto, sonido y video; permitir formas para controlar la concurrencia en las comunicaciones.

- *Mecanismos de resolución de conflictos*: Asegurar medios para expresar la diversidad de opiniones y hacer propuestas; proveer formas para visualizar el espacio de negociación con todos sus elementos y relaciones; proveer distintas herramientas para soportar decisiones.

- *Conocimiento del grupo*: Proveer herramientas para permitir, a cada participante, identificar a los miembros del grupo, y percibir cada movimiento de un miembro del grupo dentro o fuera de la ejecución de un proceso; permitir formas para reconocer contribuciones individuales y mostrar la evolución de los pasos en el proyecto durante el proceso de trabajo en equipo.

- *Definiciones prioritarias*: Dar herramientas que permitan la definición de accesos prioritarios, sus controles y su cambio dinámico, basados en las reglas de interacción del grupo.

2.- ***Procesos del grupo***:

- *Proveer acceso, uso y compartimiento de información*: Ofrecer distintas herramientas para recuperar datos en forma sincrónica y asincrónica, basados en el control de acceso especificado, que garanticen integridad de información después de la modificación de datos por miembros del grupo; y proveer mecanismos para el control de versiones.

- *Interconectar información*: Soportar distintos tipos de links para el grupo de información generado.

- *Mecanismos para búsqueda y acceso de información*: Tener formas para hacer búsquedas de manera estructurada o libre.

- *Interfaces amigables*: Si un miembro de un grupo lo pide, tener formas para controlar acciones invisibles de personas, haciéndolas visibles; administración del espacio en pantalla para evitar confusiones visuales causadas por sobrecarga de información.

- *Aceptación de firmas*: Aceptar varias formas de identificar a un usuario cuando es importante reconocer al autor de una aplicación.

3.- ***Seguridad del grupo:***

- *Control de acceso*: Soportar varios mecanismos de protección de datos; formas para permitir autorizaciones de acceso selectivas.
- *Privacidad del grupo*: Procedimientos para definir límites y hábitos del grupo a fin de observar la presencia de intrusos.

4.- ***Privacidad individual:***

- *Privacidad en el espacio de trabajo individual*: Tener medios para desconectarse del grupo temporariamente; garantizar un espacio de acciones y de datos individual; permitir la definición de puertas de acceso para su espacio de trabajo, equivalente a los mecanismos de control de acceso del sistema.
- *Transferir parte de su espacio privado*: Tener formas simples para transferir resultados de acciones e informaciones de sus espacios privados a sus espacios compartidos.

• ***Requerimientos para sistemas de computación distribuidos:***

- Seguridad y confiabilidad en la transmisión de datos
- Varios protocolos de comunicación entre distintas plataformas y aplicaciones
- Distintos enlaces de aplicaciones
- Tolerancia por fallas de usuario
- Fácil retroceso de acciones y recuperación de datos
- Facilidad para salir del contexto de trabajo
- Evolución simultánea del sistema y del usuario [8]

Capítulo 2

Distintas Aplicaciones de la Colaboración

La computación colaborativa puede ser útil en distintas disciplinas, entre ellas la escuela, la investigación, la empresa, etc.

2.1.- Colaboración en la educación

Se esta produciendo una revolución en la educación, que tiene que ver con la filosofía de cómo se enseña, con la relación maestro-estudiante, con la forma en la cual se estructura una clase y con la naturaleza del programa. En el centro de todo esto está la pedagogía.

Las características básicas pueden describirse con las palabras claves *constructivismo*, *centrada en el alumno* y *basada en problemas*. La idea es que las personas aprendan mejor cuando investigan un tema, motivadas a buscar nuevos conocimientos y habilidades para solucionar el problema que tienen entre manos. El objetivo es la exploración activa, la construcción y el aprendizaje, más que la pasividad de la lectura de texto.

En el pasado, lo más importante era el contenido; el programa estaba estructurado de acuerdo a temas básicos de literatura, historia, geografía, matemática, etc. Los expertos dividían cada área en temas más chicos y manejables y enseñaban ésto según un plan preestablecido. Ahora, la idea es la educación *centrada en el alumno*. Aquí, lo más importante son las necesidades, habilidades e intereses del alumno. Este tipo de educación, generalmente está acompañada por el tipo de educación *basada en problemas*, el cual se usa para satisfacer las necesidades e intereses de los alumnos.

Esta filosofía no es nueva, pero sí su aplicación. En el corazón del cambio aparecen nuevas tecnologías que son las que hicieron posible dicho cambio. Esta tecnología abarca el uso de fichas, videos, etc. Pero es la computadora el medio más prometedor. La combinación de aprendizaje activo y tecnología está creando situaciones de aprendizaje que no estaban disponibles hasta hace muy poco. El aprendizaje colaborativo es el que está ganando más popularidad. [10]

Los términos usados más comúnmente para describir el aprendizaje colaborativo incluyen *colaboración peer* (Tudge y Rogoff, 1984), *aprendizaje colaborativo* (Forman y Cazden 1985), *aprendizaje coordinado* (Koshmann 1994) y *aprendizaje colectivo* (Pea 1994). Pea sostiene que "no todo aprendizaje es probablemente colaborativo, algunas veces es competitivo o coercivo por naturaleza".

Forman y Cazden distinguieron el aprendizaje colaborativo de otras formas de interacción peer, basadas en la tarea que se está ejecutando. Para ello, "la colaboración requiere de una tarea mutua, en la cual los trabajadores juntos producen algo que no lo podrían producir solos".

La investigación sobre el aprendizaje colaborativo que se enfoca más sobre el proceso que sobre el resultado, es reciente y limitada. Esta mira los patrones de interacción social y las estrategias de resolución de problemas, durante la resolución de ellos por medio de grupos. Parte de los resultados muestran que, las interacciones más cooperativas que usan la mayor cantidad de combinaciones de estrategias posibles, resuelven la mayoría de los problemas. Aunque las investigaciones sobre colaboración con peers generalmente indican un avance, Tudge mostró que en ciertas circunstancias la interacción peer podría resultar tanto en desarrollo como en regresión. [11]

Los problemas auténticos típicamente requieren un amplio rango de conocimiento y de habilidades para llegar a una solución exitosa y fomentar la transferencia, demostrando cuándo es útil el conocimiento. Pero, los problemas auténticos suelen ser complicados. Los estudiantes necesitan de dos clases de ayuda para aprender:

- *Soporte para la performance del estudiante*: Si los estudiantes no perciben que pueden solucionar problemas auténticos, disminuye la motivación. El aprendizaje ocurre sólo si los estudiantes tienen la oportunidad de tener éxito. El soporte para la performance del estudiante provee la oportunidad de éxito.

- *Soporte para el aprendizaje del estudiante*: La experiencia sola no lleva al aprendizaje. El soporte para aprendizaje fomenta la reflexión sobre la experiencia.

La colaboración ayuda a la solución de problemas ya que facilita el éxito en la performance y en la reflexión sobre el aprendizaje. Los grupos pueden solucionar problemas más interesantes y complejos que el trabajo aislado de una sola persona. Los estudiantes que trabajan en grupos necesitan articular diseños, críticas y argumentos para dárselos a otros miembros del grupo, dando impulso a la clase de reflexión que lleva al aprendizaje.

En la educación, en general, se pueden apreciar tres dimensiones: compromiso, efectividad y viabilidad. Los estilos de educación *basado en problemas* y *centrado en el alumno* son los más efectivos para lograrlas:

- *Compromiso*: Un estudiante comprometido es un estudiante motivado. La motivación puede marcar la diferencia entre el éxito y el fracaso de una tarea más que cualquier otro factor. La multimedia interactiva puede motivar a los estudiantes proveyéndoles información en forma concreta y perceptiva, facilitándoles así el proceso. El compromiso tiene que ver también con la elección del tema a tratar, y, una de las cosas más importantes en la educación *basada en problemas* es usar al problema como principal fuerza de motivación.

- *Efectividad*: Lo más importante para los métodos de educación tradicional era la efectividad, sinónimo a: ¿cuánto aprendió el alumno?. Después de todo, si no hay aprendizaje del tema en cuestión, no importa cúan comprometido esté el alumno ni cúan viable sea el método, dicho método es de poco valor. Con el nuevo estilo de educación, las medidas de efectividad tradicionales basadas en tests con puntajes no serían apropiadas. Los tests tradicionales miden

conocimiento declarativo: recitaciones de un tema aprendido y aplicaciones a pequeños problemas. Esto no indica necesariamente que el alumno haya entendido el tema profundamente, ni refleja las habilidades adquiridas.

- *Viabilidad*: La demostración es comprometida, efectiva y forzada, pero... ¿es viable?. Quizás la tecnología nunca soporte a la práctica o el costo sea muy elevado. Pero, ¿qué hay con respecto a la infraestructura social y cultural que se requiere para hacer este trabajo?. La integración a los salones de clase existentes, de las herramientas de autoría, las herramientas de diseño, los componentes de software standards, la mejoría de infraestructura de distribución, etc., es crítica. Esto no es viable fuera de los salones de clase pilotos.

Para enseñar y aprender no hay un método único que pruebe ser el mejor y que solucione los problemas de educación en todos los dominios y con todo tipo de estudiantes. De hecho, se necesitan distintos métodos de enseñanza y distintos materiales para distintos estudiantes, por lo tanto el método tradicional también tiene su lugar.

Por ejemplo, la lectura de libros es aún la forma más común que se usa para presentar una gran cantidad de información secuencial en forma rápida y eficiente. Pero rotar el aprendizaje, ejercitar y practicar son tareas valiosas para la transformación del entendimiento en habilidades automatizadas, logrando información y el uso de los procedimientos disponibles, sin un esfuerzo consciente; pero son tareas débiles en cuanto a generar motivación (compromiso) y en cuanto a proveer puntos conceptuales para la comprensión.

2.2.- Colaboración en la empresa

Las organizaciones de hoy en día enfrentan un mercado muy complejo y competitivo. Para continuar siendo competitivas, estas organizaciones están buscando formas mejores de integrar los esfuerzos individuales para aumentar la productividad del grupo. Se buscan productos de software de computación colaborativa para incrementar la productividad del grupo, pues las organizaciones quieren lograr soluciones poderosas para satisfacer las necesidades de los usuarios finales y gerentes.

La computación colaborativa provee la mejor clase de productos construídos sobre sus propios componentes y la habilidad de crear soluciones al cliente construidas sobre los mismos componentes. Debido a que éstos pueden ser modificados y actualizados individualmente, un ambiente de computación colaborativa habilita la adopción de nuevas tecnologías más rápidamente, permitiendo así una solución completamente colaborativa, la cual es funcional y eficiente. Finalmente, la computación colaborativa es fácil de manejar debido a su integración con los servicios de red y a su modelo de administración centralizada.

La habilidad de setear reglas para el correo electrónico ayuda a que los mensajes sean leídos a tiempo y a que no necesiten ser borrados del sistema. El software para construír formularios hace posible, no sólo llenar una boleta rápidamente, sino que también permite al originador mantener la pista de donde está el formulario dentro del

ciclo de aprobación, quién lo ha aprobado, quién no y quién lo tiene en este momento. Los calendarios permiten a los supervisores chequear todo su calendario de reportes y rápidamente organizar una reunión a la que todos puedan concurrir. La computación colaborativa permite a las personas manejar el volumen de la información que reciben, les hace posible hacer un mejor uso de esa información, provee de conocimiento a los trabajadores y le brinda a la organización un lado competitivo, que es crucial en estos días. [12]

Los beneficios de la computación colaborativa son:

- *Reducción en tiempo de mercado*: La computación colaborativa puede tener un impacto positivo en la maduración y efectividad de las comunicaciones durante el diseño del producto y puede llevar a una reducción sustancial del tiempo de diseño. Además, puede hacer una diferencia significativa en la cantidad de tiempo que los miembros del grupo pasan asimilando información.

- *Costos operativos reducidos*: En un ambiente de manufacturación, la información puede estar distribuida en diferentes sistemas de información, requiriéndose de expertos y de tiempo para juntarla. Con el desarrollo de la computación colaborativa, la necesidad de viajar se puede reducir considerablemente. La distancia geográfica ya no importa.

- *Aumento en la satisfacción del cliente* porque se reducen los problemas de campo y el tiempo de resolución de los mismos.

Las inversiones que se requieren en tecnología para la computación colaborativa varían de compañía a compañía, dependiendo de: la cantidad de lugares y el número de usuarios involucrados, y la necesidad de instalación de redes o mejoras.

Los sistemas de computación pueden definirse como listos para la colaboración en la medida que soporten el uso de sonido, video, imágenes y modelos 3D. La prioridad y performance requerida para cada uno de ellos, variará de una organización a otra. Estas prioridades deben determinarse antes de determinar qué nivel de soporte y qué mejoras pueden requerirse para las estaciones de trabajo y servers involucrados. Una vez que estas prioridades fueron establecidas, se deben analizar el hardware y el software asociados y llevarlo a una base de performance y funcionalidad.

El entrenamiento es una inversión crítica si la compañía quiere aprovechar al máximo los beneficios de la computación colaborativa lo antes posible. Lo más efectivo sería:

- Una clase de entrenamiento formal con un tiempo de laboratorio adecuado, cubriendo todas las herramientas a usarse.

- La presencia de expertos en cada grupo, que puedan proveer asistencia sobre la marcha con respecto al uso de las herramientas.

- Encuentros regulares en los cuales los usuarios puedan compartir ideas, problemas y éxitos.

Si la computación colaborativa se transformara en parte del conjunto principal de las herramientas de una compañía, los usuarios deberían tener tiempo para "jugar" y para descubrir la comodidad de dichas herramientas. De otra manera, la presión por tener el trabajo terminado puede causar que los usuarios vuelvan a su forma de trabajo tradicional.

El desarrollo de la tecnología de la computación colaborativa requiere cooperación y coordinación entre una variedad de grupos y lugares:

- Los participantes necesitan ser manejados al mismo ritmo, sin importar las habilidades de la computadora ni los niveles de sofisticación. Si no es así, ellos no serán capaces de colaborar en tareas críticas cuando sea necesario.

- Las organizaciones elegidas para el desarrollo deben tener la oportunidad de usar las herramientas para interactuar con otros grupos de individuos a fin de completar exitosamente sus objetivos y tareas.

- Los individuos y los grupos necesitan tener las ideas claras en cuanto a cómo aplicar la tecnología colaborativa en sus trabajos.

- Se debe asegurar que los objetivos individuales y grupales estén al mismo nivel en el desarrollo de la tecnología colaborativa.

- La tecnología colaborativa puede facilitar las comunicaciones, pero no reparará las malas relaciones que ya existan entre grupos o individuos. [13]

Capítulo 3

Toma de decisiones en grupo

3.1.- El trabajo en grupo

Hay muchas ventajas cuando se trabaja en grupo, en un proyecto:

- Aprender a planear y organizar.
- Aprender a negociar y comprometerse.
- Practicar habilidades de toma de decisiones.
- Conocer nuevas personas.
- Aprender de otras personas.
- Que cada individuo trabaje menos.
- Divertirse.

El primer paso del trabajo en grupo es la elección de los integrantes del mismo. Muchas personas intentan agruparse con sus mejores amigos, pero ésta no es siempre la mejor elección, ya que se deben tener en cuenta ciertas características como intereses compartidos, tiempo disponible, hábitos de trabajo, conocimiento, resultados esperados, etc.

El próximo paso sería conocer más a fondo a los integrantes del grupo, para lo cual es conveniente intercambiar números de teléfono, direcciones, etc.; establecer un método de comunicación (mail electrónico, teléfono, visitas); acordar pautas sobre el funcionamiento del grupo.

Luego, se debe elegir el tema y determinar su alcance. El proyecto debe ser apropiado para todos los miembros del grupo. Cada miembro tendrá derecho a tomar decisiones respecto a problemas relacionados con su propio trabajo, por ejemplo, si a Juan le gusta el color azul y decide usarlo en un gráfico, entonces Pedro no tiene derecho a cuestionar esta decisión (a menos que viole una decisión mutua previa). Además, cada miembro del grupo tendrá ciertas responsabilidades:

- Encontrar las interfaces convenidas, a menos que primero notifique a sus compañeros de sus intenciones de cambio.
- Encontrar los lineamientos establecidos mutuamente.
- Investigar y elegir el mejor método para implementar la tarea que le fue asignada. [21]

3.2.- Toma de decisiones

La toma de decisiones en grupo es el proceso de llegar a un resultado basado en las opiniones de muchos individuos. Además, es un componente clave en el funcionamiento de un grupo, debido a que la performance del grupo involucra mucho más que sólo acciones individuales.

El proceso de toma de decisiones se caracteriza por:

- Delegación de responsabilidades y autoridad.
- Estar lo más cerca posible del proceso de identificar problemas y recursos para ayudar a resolverlos.
- Compromiso y atención tanto al proceso de toma de decisiones como al resultado.
- Recolectar entradas de quienes estarán afectados por la decisión y de quienes estarán involucrados en lograr los resultados.
- Comunicación abierta pero respetándose los tiempos a través de todos los niveles del grupo.
- Cooperación, negociación, compromiso.
- Tomar la responsabilidad individual por la entrada, el proceso, el resultado y la implementación.

La toma de decisiones compartidas implica:

- Mejores decisiones.
- Decisiones que involucren a los más cercanos al problema.
- Amplio compromiso y habilidad para implementar la decisión.
- Grupos mejor educados e informados.
- Acuerdos sobre el proceso por el cual se tomaron las decisiones.
- Entendimiento de por qué se tomó una determinada decisión.
- Libre intercambio de ideas y opiniones. [20]

3.2.1.- Modelos

[22] Debido a la importancia del proceso de toma de decisiones en grupo, se pueden usar modelos a fin de establecer medios sistemáticos para desarrollar tomas de decisiones en grupo efectivas. En general, se pueden identificar cuatro modelos de toma de decisiones en grupo, que son: racional, político, proceso y tarro de basura.

3.2.1.1.- Modelo racional

Este modelo se fundamenta en objetivos, alternativas, consecuencias y optimizaciones. El modelo asume que se dispone de información completa con respecto a la decisión que se va a tomar, a partir de la cual se podrá determinar una concepción correcta del problema. También asume que los integrantes del grupo evalúan consistentemente las ventajas y desventajas de cada alternativa teniendo en mente los objetivos. Luego, ellos evalúan las consecuencias de seleccionar o no cada alternativa. Se elegirá la que provea la mayor utilidad.

La principal ventaja del modelo racional es que utiliza un proceso secuencial y lógico. Las decisiones se toman deductivamente, determinando los objetivos a lograr, evaluando las alternativas potenciales, basadas en la información con que se dispone, y eligiendo la alternativa óptima. En otras palabras, el modelo es simple e intuitivo.

Este modelo tiene una desventaja importante. Asume que no hay predisposiciones intrínsecas en el proceso de toma de decisiones. Este optimismo puede no ser totalmente realista, ya que los individuos involucrados en el proceso traen sus

propias percepciones y modelos mentales. Además, parece que las predisposiciones intrínsecas son inevitables y deberían poder manejarse.

3.2.1.2.- Modelo político

El modelo político considera como parte del proceso de decisión, a las nociones preconcebidas que tienen los integrantes del grupo. En contraste con el modelo precedente, los individuos involucrados no realizan la tarea de decisión mediante una elección racional, de acuerdo a los objetivos, sino que están motivados y actúan en virtud a sus propias necesidades y percepciones. Este proceso involucra un ciclo de negociación entre los integrantes, cada uno pretende que sus perspectivas sean las elegidas. Más específicamente, implica que cada miembro trate de convencer, a las personas con más poder, de adoptar sus puntos de vista, y así influenciar al resto.

Además, el modelo político no involucra la disponibilidad de información completa, ni centrarse en el punto de vista óptimo. La información completa es poco deseable, ya que el modelo político se basa en negociaciones que, a menudo, están influenciadas por el poder y por favores. En realidad, a veces la información es retenida para maniobrar mejor una perspectiva dada. Debido a esto, el punto de vista óptimo no es un aspecto clave de este modelo.

La ventaja del modelo político es que provee una representación subjetiva de la forma en la cual opera el mundo real, y puede minimizar conflictos. Los individuos siempre tendrán sus propias preferencias, las cuales influenciarán en sus comportamientos. Identificando o conociendo este hecho en el proceso de toma de decisión, se pueden prevenir y minimizar los problemas potenciales y los conflictos. Los conflictos también se pueden minimizar convenciendo a las personas con poder a que se adhieran a un punto de vista particular, a fin de lograr que otros miembros del grupo, indecisos, también lo hagan.

Si bien el modelo político tiene la ventaja de emular la forma en que se comporta el mundo real, este hecho también es una desventaja, porque puede no seleccionarse la mejor solución o decisión.

3.2.1.3.- Modelo de proceso

En contraste con el modelo político, este modelo es más estructurado. Las decisiones se basan en procedimientos operativos standard o en lineamientos preestablecidos en el grupo. Las acciones y comportamientos ocurren de acuerdo a estos procedimientos o lineamientos.

Además, la organización de los eventos pasados, presentes y futuros, así como también la conformidad, son partes integrales de este modelo. La organización de eventos pasados, presentes y futuros es importante porque pueden usarse como un fundamento consistente para la toma de decisiones. Considerar estos eventos a través del tiempo provee un mayor refinamiento de los lineamientos que ayudan a determinar los resultados.

La conformidad es una parte integral del modelo de proceso porque es el medio por el cual se manejan las dudas e incertidumbres durante la tarea de decisión. Si los integrantes del grupo no están seguros con respecto a la efectividad potencial o a los resultados de una decisión, se conformarán con los standards preestablecidos. Esta conformidad no debería interpretarse como que la decisión no tiene un fundamento sólido. En este caso, la conformidad apenas se relaciona con el hecho de que el razonamiento que llevo a esa decisión estuvo basado en lineamientos predeterminados.

3.2.1.4.- Modelo tarro de residuos

Este modelo es el más apropiado para tareas de elección en grupos donde las tecnologías no son claras, varía el esfuerzo y tiempo que le dedica cada integrante y las opciones son inconsistentes y están mal definidas.

En un grupo, una oportunidad de tomar una decisión puede describirse como un tarro de residuos en el que los integrantes del grupo depositan distintos tipos de problemas y soluciones, independientemente de la forma en que se generaron. Los problemas, soluciones e integrantes no están necesariamente relacionados. Ellos cambian de una solución a otra, de tal manera que, el tiempo y los problemas parecen estar relacionados con una alineación casual de componentes que permitirá completar la decisión. Estos componentes son la combinación de las opciones disponibles en un momento dado, la combinación de problemas, la combinación de soluciones y las demandas externas de los integrantes.

La principal ventaja de este modelo es que provee una representación del mundo real de la forma no racional en que a menudo se toman las decisiones dentro de un grupo. No todas las decisiones se toman en forma lógica, política o standard. Ocasionalmente, se deciden en forma ad-hoc cuando las soluciones, problemas e individuos involucrados en la tarea se ponen de acuerdo.

A pesar de que el modelo de tarro de residuos representa a las decisiones en una forma real y no racional, éste tiene una importante desventaja. No es el medio más eficiente para tomar decisiones. La toma de decisiones se considera un procedimiento para encontrar soluciones a problemas. Desafortunadamente, esto no ocurre si las decisiones se toman usando el modelo tarro de residuos, ya que si bien los problemas se analizan en determinadas situaciones, las elecciones se hacen únicamente cuando la combinación de problemas, soluciones e individuos permite la toma de decisiones. Como consecuencia, la lineación de problemas, soluciones e individuos muchas veces ocurre una vez que ya paso la oportunidad de tomar una decisión con respecto a un problema, u ocurre antes de que el problema se haya descubierto.

3.2.2.- Fases

La mayoría de los modelos de toma de decisiones incluyen, al menos, cuatro fases:

- *Fase de entrada:* se percibe un problema y se intenta entenderlo o entender la situación.
- *Fase de procesamiento:* se generan y evalúan alternativas, y se selecciona una solución.

- *Fase de salida:* se planea e implementa la solución.
- *Fase de revisión:* se evalúa la solución y se le hace modificaciones, si es necesario.

El proceso de toma de decisiones comienza con la percepción de un gap entre una situación actual y un objetivo deseado, con el camino hacia dicho objetivo bloqueado por obstáculos conocidos o desconocidos. En general, esta situación no se ha presentado anteriormente o no se conoce una solución específica a partir de las experiencias pasadas.

3.2.2.1.- Fase de entrada

El objetivo de esta fase es entender mejor el problema o situación. El primer paso es identificar el problema y exponerlo clara y concisamente. Identificar el problema significa describir, lo más precisamente posible, el gap entre lo que uno percibe en las circunstancias presentes y lo que a uno le gustaría que ocurra. La identificación del problema es vital para comunicarse unos a otros, el centro del proceso de toma de decisión.

El segundo paso de esta fase de entrada es establecer el criterio que se va a usar para evaluar las posibles alternativas del problema y la efectividad de las soluciones seleccionadas. Durante este paso es importante establecer cualquier límite identificado sobre alternativas aceptables, valores importantes o sentimientos a ser considerados, o resultados que deberían evitarse. Además, se deberían categorizar criterios como esenciales para una solución exitosa o apenas deseables.

El tercer paso es recolectar información o hechos relevantes para la toma de una decisión. Este paso es crítico para el entendimiento de las condiciones iniciales y para una posterior clarificación del gap percibido. La calidad de los hechos es más importante que la cantidad.

3.2.2.2.- Fase de procesamiento

En la fase de procesamiento la tarea es desarrollar, evaluar y seleccionar alternativas y soluciones que puedan resolver el problema. El primer paso en esta fase es desarrollar alternativas o posibles soluciones. Esta generación debería ser libre, abierta e indiferente con respecto a la factibilidad. Se debería emplear el tiempo suficiente en esta actividad de manera de asegurar la generación de alternativas no standard y creativas.

El próximo paso es evaluar las alternativas generadas de acuerdo al criterio establecido. Cada integrante del grupo escribe individualmente las ventajas, desventajas y aspectos interesantes de cada alternativa, y luego las comparte y discute con el grupo. Después de descartar las alternativas que claramente están fuera de los límites de los criterios acordados previamente, se deberían considerar en más detalle las ventajas y desventajas. También, se debería realizar un análisis de las relaciones que hay entre las alternativas y considerar la importancia relativa de las ventajas y desventajas. Sólo se usarán más adelante, aquellas alternativas que la mayoría considere relevantes y correcta.

El tercer paso es desarrollar una solución que resuelva exitosamente el problema. Para problemas relativamente simples, una de las alternativas puede ser obviamente superior. Sin embargo, en situaciones complejas es preferible combinar varias alternativas para lograr una solución más efectiva.

Antes de dejar esta fase, es importante diagnosticar los posibles problemas que se pueden presentar con la solución y las implicancias de ellos. Cuando se desarrolla una solución es importante considerar lo peor que pueda ocurrir cuando ella se implemente.

3.2.2.3.- Fase de salida

Durante esta fase se desarrolla un plan y se implementa la solución. El plan debe ser lo suficientemente detallado como para permitir una implementación exitosa; también se deben considerar y desarrollar métodos de evaluación. Cuando se desarrolla un plan, se consideran primero las fases principales y luego los pasos necesarios para cada fase. A menudo es útil construír un diagrama de tiempo y hechos con los pasos más importantes de la implementación. Luego, se implementa el plan tan cuidadosa y completamente como sea posible, siguiendo los pasos que se desarrollaron y haciendo la menor cantidad de modificaciones posibles.

3.2.2.4.- Fase de revisión

Este paso, la evaluación de la implementación de la solución, debería ser un proceso dinámico. Se debe considerar la completitud de la implementación antes de evaluar su efectividad. Este paso muchas veces es omitido y es una de las razones por las que puede fallar el proceso de toma de decisiones: la solución seleccionada no ha sido implementada efectivamente. Sin embargo, si la solución no se implementó, entonces la evaluación de efectividad no será válida.

El segundo paso de esta fase es evaluar la efectividad de la solución. Es importante evaluar los resultados teniendo en cuenta los problemas que se pensaron en el comienzo de proceso. Se deberían considerar los resultados afectivos, cognitivos y de comportamiento, especialmente si habían sido identificados como criterios importantes. La solución debería ser juzgada de acuerdo a su eficiencia, su impacto sobre las personas involucradas y su alcance, el cual es valorado por los participantes.

El paso final en el proceso es modificar la solución en la forma sugerida por el proceso de evaluación.

3.2.3.- Estrategias para la toma de decisiones

Ante cualquier elección que tengan que hacer los miembros del grupo, en cualquiera de las fases de cualquiera de los modelos, es preciso tomar decisiones. Generalmente, no todos los miembros del grupo estarán de acuerdo el cien porciento de las veces. Existen tres estrategias de toma de decisiones para cuando los miembros del grupo están en desacuerdo:

- *Decisión por mayoría:* Se realiza una votación y gana la mayoría. Esta estrategia obviamente no funcionará para grupos de dos personas. Más aún,

en una decisión por mayoría, los perdedores no estarán de acuerdo con la elección y podrían tratar de sabotear el proyecto.

- *Decisión por concenso unánime:* Ocurre cuando todos están verdaderamente de acuerdo con respecto al curso de acción a tomar. Si se logra el concenso unánime, es ideal. Sin embargo, lograrlo puede requerir una inversión de tiempo demasiado importante.

- *Decisión por concenso:* Ocurre cuando cada integrante del grupo siente que tiene una buena chance de influenciar en la decisión. Cada persona no necesariamente está de acuerdo con la decisión, pero entiende la decisión final y está dispuesta a apoyarla. El concenso generalmente se logra por compromiso. Cuando hay un desacuerdo, ningún miembro debería esperar hacer todo a su manera. Cada miembro debe decidir qué es lo más importante y aceptar las ideas de los demás. La toma de decisiones por concenso es la más recomendable. Es más eficiente que la decisión por concentimiento unánime y es mucho más probable que resulte en la satisfacción completa de todos los miembros, que las decisiones tomadas por mayoría.

Capítulo 4

Internet

[23] Internet ... una red de redes basada en protocolos TCP/IP, una comunidad de personas que usan y desarrollan esas redes, una colección de recursos que se pueden alcanzar con ellas.

4.1.- Introducción

Una pregunta muy común es: ¿qué es Internet ? La razón por la cual esa pregunta se hace tan frecuentemente es porque no hay un acuerdo sobre una respuesta que refleje nítidamente Internet. Se puede pensarse a Internet como una relación con sus protocolos comunes, como una colección física de routers y circuitos, como un conjunto de recursos compartidos o aún como una actitud sobre interconección e intercomunicación.

Hoy, Internet es un recurso global que conecta a millones de usuarios que comenzaron hace 20 años atrás como un experimento, a partir del Departamento de Defensa de los Estados Unidos. Aunque las redes que componen Internet estén basadas en un conjunto de protocolos standard, Internet también tiene gateways a redes y servicios basados en otros protocolos.

4.2.- Historia

Internet nació más o menos 20 años atrás, en un intento de conectar la red del Departamento de Defensa de los Estados Unidos llamada ARPAnet con otras redes de radio y satélite. La ARPAnet fue una red experimental diseñada para soportar investigaciones militares, en particular, investigaciones sobre cómo construír redes que pudieran soportar outages parciales (como bombardeo) y que aún sigan funcionando. En el modelo de la ARPAnet, las comunicaciones siempre se producían entre una fuente y la computadora destino. La red misma asumía que no era confiable; cualquier porción de la red podía desaparecer en cualquier momento. Esto fue diseñado así para que se requiera de un mínimo de información de las computadoras cliente. Para enviar un mensaje por la red, una computadora sólo tenía que poner sus datos en un sobre, llamar a un paquete del Protocolo de Internet (IP), y direccionar los paquetes correctamente. Las computadoras comunicadas tenían también la responsabilidad de asegurar que la comunicación se completara. La filosofía era que cada computadora de la red, pudiera hablar de igual a igual (como un peer), con otra computadora.

Aunque la Organización para la Standarización Internacional (ISO) estuvo, por años, tratando de diseñar un standard para las redes de computadoras, las personas no podían esperar. Los desarrolladores de Internet en los Estados Unidos, Reino Unido y Escandinavia, respondiendo a las presiones del mercado, comenzaron a poner su software IP en cada tipo concebible de computadora. Esto se transformó en el único medio práctico que tenían para comunicarse las computadoras hechas por diferentes fabricantes.

Más o menos al mismo tiempo, se desarrollaron las redes locales Ethernet (LANs). Esta tecnología maduró tranquilamente, hasta que estuvo disponible para las estaciones de trabajo en 1983. La mayoría de estas estaciones venían con el sistema Berkeley UNIX, el cual incluía el software de red IP. Al mismo tiempo, otras organizaciones empezaron a construír sus propias redes usando los mismos protocolos de comunicación que la ARPAnet. Era obvio que si estas redes podían hablar juntas, los usuarios de una red podían comunicarse con los usuarios de otras, así, todos se beneficiaban.

Una de las cosas más importantes de estas nuevas redes era el NSFNET, comisionado por la Fundación Nacional de Ciencias (NSF), una agencia del Gobierno de los Estados Unidos. En la última parte de los '80, el NSF creó cinco centros de super computadoras, fomentando a que los recursos estuvieran disponibles para cualquier investigación escolar. Sólo se crearon cinco centros porque eran muy costosos, por lo tanto deberían compartirse. Esto creó un problema de comunicaciones: se necesitaba una forma de conectar todos los centros entre sí y permitirle a los clientes de estos centros, acceder a ellos. Al principio, el NSF trató de usar ARPAnet para las comunicaciones, pero esta estrategia falló por la burocracia y los problemas de plana mayor (staff).

En respuesta, el NSF decidió construír su propia red, basada en la tecnología IP de la ARPAnet. Esta conectó los centros a una velocidad de 56.000 bits por segundo que es lo que permitían las líneas telefónicas. Era obvio, sin embargo, que si se trataba de conectar cada universidad directamente al centro de supercomputadoras, se paraba todo. Entonces, se decidió crear redes regionales. En cada área del país, las escuelas podrían conectarse a su vecino más cercano. Cada cadena estaría conectada a un centro de supercomputadoras en un punto, y a su vez, los centros estarían conectados entre sí. Con esta configuración, cualquier computadora podría, eventualmente, comunicarse con cualquier otra, enviando su conversación a través de sus vecinos.

Esta solución tuvo éxito. Compartir las supercomputadoras también permitió a los sitios conectados, compartir muchas otras cosas no relacionadas con los centros. El tráfico de la red aumentó hasta que, eventualmente, las computadoras que controlaban la red y las líneas telefónicas fueron sobrecargadas. En 1987, se firmó un contrato con Meric para que manejara y mejorara la red.

La vieja red fue reemplazada por líneas telefónicas más rápidas, con computadoras más rápidas para controlarlas.

El proceso de quedarse sin caballos de fuerza y conseguir motores más grandes y mejores carreteras, sigue hasta nuestros días. Sin embargo, la mayoría de estos cambios no son notados por las personas que tratan de usar Internet para trabajar. El proceso de que se agote la capacidad y se mejore entonces la red, ha creado una tecnología que es extremadamente madura y práctica. Las ideas fueron testeadas; aparecieron problemas pero fueron solucionados.

Para nuestros propósitos, el aspecto más importante de los esfuerzos con respecto a la red del NSF fue el hecho de permitir a todos acceder a la red. En este punto, el acceso a Internet estaba disponible sólo para los investigadores de la ciencia de

la computación, y, para empleados y contratados del gobierno. El NSF promovió el acceso de todos los sectores de la educación fundando conecciones con los campus sólo si el campus tenía planeado distribuír ese acceso.

La demanda comenzó a crecer. La gente estaba tratando de conseguir que los colegios primarios y secundarios pudieran conectarse. Los graduados conocían para qué servía Internet, y le hablaron a sus empleadores para conectar sus corporaciones. Toda esta actividad apuntó al continuo crecimiento, a solucionar los problemas de redes, a la evolución de las tecnologías y a una seguridad de trabajo para los encargados de la red.

4.3.- ¿Qué la compone?

Qúe compone Internet es una pregunta difícil; la respuesta cambia con el tiempo. Cinco años atrás, la respuesta hubiera sido fácil: "Todas la redes que usaran el protocolo IP, podían cooperar para formar una red compacta para sus usuarios colaborativos".

Más recientemente, algunas redes no basadas en IP dijeron que Internet era buena. Y querían proveer servicios a su clientela. Así que desarrollaron métodos para conectar estas redes "extrañas" a Internet. Al principio estas conecciones, llamadas gateways, apenas sirvieron para transferir correos electrónicos entre dos redes. Algunas, sin embargo, han crecido de manera de transportar por la red otros servicios. ¿Son ellas parte de Internet? Quizás sí o quizás no. Depende de si, en sus corazones, quieren serlo o no.

4.4.- ¿Quién la gobierna?

En muchas formas, Internet es como una Iglesia: tiene un consejo de ancianos, cada miembro tiene una opinión sobre las cosas en las que se debería trabajar, y uno puede tomar parte o no. Internet no tiene un presidente, un jefe de operaciones o un Papa. Las redes que la constituyen pueden tener presidentes, pero esto es una característica diferente; no hay una sola figura de autoridad para Internet como un todo.

La autoridad principal que decide los destinos de Internet es la Sociedad de Internet (ISOC). ISOC es una organización con miembros voluntarios cuyo propósito es promover el intercambio de información global a través de la tecnología de Internet. Esto es similar a un concilio de ancianos, quienes tienen la responsabilidad del manejo técnico y la dirección de Internet.

El concilio de ancianos es un grupo de invitados voluntarios llamados el Equipo de Arquitectura de Internet (IAB=Internet Architecture Board). El IAB se reúne regularmente para "bendecir" a los standards y para alocar recursos, como direcciones. Internet trabaja porque hay formas standard de computadoras y aplicaciones de software que les permiten hablar entre ellas. Esto hace que las computadoras de distintas marcas se comuniquen sin problemas. El IAB es responsable por estos standards; decide cuándo se necesita uno, y cuál puede ser. Cuando se requiere de un standard, se considera el problema, se adopta un standard, y se lo anuncia vía la red. Por ejemplo, cada computadora sobre Internet tiene direcciones únicas de 32 bits; ninguna otra computadora tiene la misma dirección. ¿Cómo se asigna esta dirección? El IAB se

preocupa de este tipo de problemas. Este no asigna las direcciones, sino que hace las reglas sobre cómo deben asignarse.

Los usuarios de Internet expresan sus opiniones a través de reuniones de la Fuerza de Tareas de Ingeniería de Internet (IETF= Internet Engineering Task Force). La IETF es otra organización voluntaria; se reúne regularmente para discutir problemas técnicos operacionales y de corto plazo. Cuando considera a un problema lo suficientemente importante para merecer una atención especial, el IETF organiza un grupo de trabajo para una investigación detallada. Los grupos de trabajo pueden tener diferentes funciones, variando desde la producción de documentación, hasta decidir qué redes pueden cooperar cuando ocurren problemas o hasta cambiar el significado de los bits en ciertos tipos de paquetes. Un grupo de trabajo generalmente produce un informe. Dependiendo de la clase de recomendación, ésta puede transformarse en una documentación que se reparte a todos los que la quieran, puede ser aceptada voluntariamente como una buena idea que seguirán las personas, o puede ser enviada al IAB para que sea declarada como standard.

Así es Internet. Si una red acepta las enseñanzas de Internet, se conecta a ella y se considera como parte de ella, entonces esa red es parte de Internet. Puede llegar a encontrar cosas que no le gustan y puede mandar estos comentarios al IETF. Algunas inquietudes pueden considerarse válidas e Internet puede cambiar de acuerdo a ellas. Algunos de los cambios pueden ser rechazados. Si la red hace algo que causa daño a Internet, puede ser descomunicada hasta que solucione sus malas costumbres.

4.5.- ¿Quién paga?

Nadie paga por esto. No existe una Internet, Inc. que recolecte pagos de todas las redes o usuarios de Internet. En su lugar, cada uno paga su parte. El NSF paga por el NSFNET. La NASA paga a la NASA Science Internet. Las redes se fueron uniendo y decidieron cómo conectarse entre ellas y fundaron así estas interconecciones. Un colegio o corporación paga por su conección, a alguna red regional, la cual a su vez paga al proveedor nacional de ese acceso.

4.6.- ¿Qué significa para el usuario?

El concepto de que Internet no es una red, sino una colección de redes, significa poco para el usuario final. El usuario quiere hacer algo útil: correr un programa, o acceder a algún dato único. El usuario no debe preocuparse por cómo está unido todo esto. Debe tener en cuenta que el sistema telefónico, también es una Internet. Pacific Bell, AT&T, MCI, etc. son todas corporaciones separadas que corren piezas del sistemas telefónico. Ellos se preocupan de que todo trabaje en conjunto, y el usuario sólo tiene que discar.

Cada red tiene su propio centro de operaciones de red (NOC). Los centros de operaciones se comunican unos con otros y saben cómo resolver los problemas. Cada lugar tiene un contrato con una de las redes que constituyen Internet, y el trabajo del contratante es mantener feliz a su lugar. Así que si algo va mal, ellos son los que se encargan. Si no hay problemas, son pasados por alto.

4.7.- ¿Qué se trae el futuro ?

El futuro de las redes tiene que ver con lo siguiente:

• *Nuevos protocolos standard:* La Organización Internacional de Standards (ISO) finalmente terminó de diseñar un conjunto de protocolos standard, por lo tanto ahora existe un standard internacional, que se refiere típicamente a cómo es apropiado el protocolo para ISO/OSI (Open System Interconnect). Muchas de las redes componentes de Internet permiten usar OSI hoy. Pero no hay mucha demanda.

Muchas personas sienten que la actual aproximación funciona bien, así que, ¿por qué eliminarla?. Estas personas se sienten cómodas con lo que tienen, ¿por qué deberían aprender un nuevo conjunto de comandos y terminologías sólo porque son standard ?

Actualmente no hay ventajas reales en cambiarse a OSI. Es más complicado y está menos maduro que IP, y por lo tanto no trabaja tan eficientemente. OSI ofrece la esperanza de nuevas características, pero todavía sufre de algunos de los mismos problemas que invadieron a IP hasta que la red se hizo más fuerte y rápida. Es claro que algunos lugares se convertirán a los protocolos OSI dentro de algunos años. La pregunta es: ¿cuántos?.

• *Conecciones Internacionales*: Internet ha sido una red internacional por largo tiempo, pero sólo se había extendido a los países aliados de Estados Unidos y a las bases militares del otro lado del mar. Ahora, Internet se está esparciendo por todos lados. Está en más de 50 países y el número va creciendo rápidamente.

En este momento, la expansión internacional de Internet está detenida por la falta de una buena infraestructura de soporte, digamos un sistema telefónico decente. Tanto en Europa oriental como en el tercer mundo, el estado del arte de los sistemas telefónicos es inexistente. Aún en ciudades más grandes, las conecciones están limitadas a las velocidades disponibles, que son las de las casas promedio de los Estados Unidos, 9.600 bits por segundo. Típicamente, aún si estos países están en Internet, sólo podrán acceder a algunos lugares. Sin embargo, como los sistemas telefónicos mejoran cada día, se puede esperar que esto también cambie; cada vez más se verán sitios pequeños conectados a Internet.

• *Comercialización:* Muchas grandes corporaciones han estado conectadas a Internet por años. En la mayoría de los casos, su participación ha sido limitada a sus departamentos de investigación e ingeniería. Estas mismas corporaciones usaron otras redes para sus comunicaciones comerciales.

Los negocios ahora descubrieron que usar muchas redes es muy costoso. Algunas están empezando a mirar a Internet como un elemento comercial de red. En el pasado, ellos se asustaron por las políticas que excluían o restringían su uso comercial. Muchas de estas políticas están bajo revisión y van a cambiar. Cuando se acaben estas restricciones, el uso comercial de Internet comenzará a ser más común, progresivamente.

• *Privatización*: Junto con la comercialización viene la privatización. Por años, la comunidad de redes ha querido que las compañías telefónicas les provean conecciones IP. Esto es, de la misma manera que se puede ordenar un línea telefónica para poner un teléfono en una casa, se podría hacer lo mismo con la conección a Internet. Se hace el

pedido, el instalador de teléfonos hace su trabajo y se va, y usted podría conectar la computadora a Internet. Las compañías telefónicas han dicho históricamente: "Nosotros les vendemos las líneas telefónicas, y ustedes pueden hacer lo que quieran con ellas". Por defecto, el gobierno federal estableció el negocio de las redes.

Ahora que las grandes corporaciones han comenzado a interesarse en Internet, las compañías de teléfonos han empezado a cambiar su actitud. Ellos y otros proveedores de redes se quejan de que el gobierno debería dejar el negocio de las redes.

Aunque la mayoría de las personas de la comunidad de las redes piensan que la privatización es una buena idea, hay algunos obstáculos en el camino. La mayoría da vueltas con respecto a qué va a pasar con las conecciones que ya existen. Muchas escuelas están conectadas porque el gobierno paga parte de la cuenta. Si ellos tuvieran que pagar por sus propios medios, muchas decidirían gastar el dinero en otras cosas. Las instituciones de investigación más importantes podrían seguir conectadas a la red; pero los colegios más pequeños no y los costos serían probablemente prohibitivos para la mayoría de las escuelas secundarias. Internet todavía no se ha transformado en una necesidad para la mayoría de las personas. Cuando lo sea, la privatización llegará pronto.

Capítulo 5

Groupware

El concepto de groupware intenta alentar la colaboración y la productividad interpersonal, automatizando muchas tareas o mejorando la eficiencia de otras.

5.1.- Introducción

La tendencia hacia la colaboración es muy fuerte, empujada desde el lado de la tecnología con la proliferación de las redes. El crecimiento de las redes, que ha sido explosivo en los últimos ocho años, está empujado por presiones económicas y organizacionales tales como incrementar la competencia global y la recesión que ha estado rondando por el mundo. Además, las compañías compiten por ser más eficientes tomando ventajas de la tecnología y se están dando cuenta de que la vieja organización jerárquica no es adecuada. Muchas compañías están en procesos de reingeniería usando herramientas de groupware.

Groupware es una tecnología colaborativa. Esto significa que impacta en la forma en que las personas se comunican. Impactar en las comunicaciones implica impactar en la forma en que la gente trabaja y eventualmente en la estructura de la organización. Groupware es una herramienta que usa la gente. La dificultad que encuentran la mayoría de las organizaciones es la relación entre la tecnología y las personas, que son quienes usan groupware.

El manejo de los cambios es un conjunto de prácticas y tecnologías que ha evolucionado más allá del campo del desarrollo organizacional. Dicho manejo es crítico en groupware. El planeamiento de los cambios mejora drásticamente la posibilidad de éxito. Adicionalmente, las organizaciones tienden a resistirse a los cambios en proporción a su tamaño. Cuanto más grande es la organización o el cambio, mayor es la resistencia. [17]

Por encima y por debajo de groupware, las tecnologías deben tener el suficiente impacto sobre la forma en que las personas trabajan y se comunican, deben magnificar el grado de cambio y deben poder engendrar opiniones más fuertes, tanto a su favor como en su contra.

Por otra parte, la palabra groupware es frecuentemente una fuente de confusión. Este estado de sensaciones es el síntoma de dos tendencias subyacentes. Primero, los proveedores, usuarios y observadores han tendido a considerar groupware como la suma de sus aplicaciones, con algo más para adecuarlas a la infraestructura y tecnología óptimas. Ha sido difícil derivar un entendimiento consistente de la tecnología de groupware a partir de aplicaciones tan dispares como: correo electrónico, inventario/calendario de grupos, ruteo de grupos, automatización del flujo de trabajos, conferencias por computadora, tablas de informes y sistemas de video-conferencias, entre otros.

Segundo, las discusiones de groupware tienden a centrarse en tecnologías con poco centro de diseño. Los proveedores construyen productos basados en las tecnologías más familiares para ellos. Así, los proveedores cuyos productos de basan en comunicación, ven al manejo de mensajes como la tecnología central de groupware. Los proveedores que se centran en colaboración ven a las conferencias por computación y a las bases de datos compartidas como el centro de groupware. Mientras que los proveedores que se dedican a asistir a individuos y a grupos en la coordinación de tareas complejas que involucran una mezcla de delegación secuencial de señales, etc., ven como el centro de groupware a las aplicaciones que soportan el manejo y la automatización del flujo de trabajos.

En realidad, la infraestructura completa de groupware, no sólo soporta estos tres modos de trabajo en grupo, sino que crea una relación muy fuerte entre ellos, produciendo un todo mucho más grande que la suma de sus partes.

Aparentemente, hay una relación natural entre comunicación, colaboración y coordinación en la evolución de los grupos y de las tecnologías que los soportan. Específicamente, las tecnologías de envíos de mensajes y de bases de datos compartidas son la base fundamental para los tres modos de trabajo en grupo existentes y la integración entre ellos: una arquitectura de bases de datos compartidas provee a los miembros de los grupos la posibilidad de diseñar, manejar y mantener procesos grupales, y el envío de mensajes considera, al transporte de información de la forma guardar y seguir, como el medio de comunicación fundamental entre personas y aplicaciones.

5.2.- Definición

Debido a que groupware es un término nuevo aplicado a diferentes tecnologías, es que todavía no hay una definición en la cual todos coincidan. Hay tantas definiciones de groupware como personas tratando de definirlo. Las tres definiciones más populares son:

- Procesos grupales intencionales más un software para soportarlos (Peter y Trudy Johnson-Lenz, 1978)
- Un sistema humano-herramienta co-evolucionario (Doug Englebart, 1988)
- Colaboración por medio de computadoras que incrementa la productividad o funcionalidad de los procesos persona a persona (David Coleman, 1992)

Más que debatir cuál es la mejor definición para groupware, nos preguntamos... ¿es realmente importante una definición? Lo que importa es si la tecnología de groupware provee una solución al problema específico de las compañías o no.

Las definiciones de groupware que van más allá de: "software que soporta el trabajo en grupos", generalmente se basan en un sólo aspecto del trabajo en grupo y en una aplicación clave que soporta esa clase de trabajo. Los usuarios de correo electrónico y de las aplicaciones habilitadas para ellos, se inclinan a ver a groupware como parte del envío de mensajes. Los usuarios de productos para ruteo de formularios piensan que groupware es una función de la automatización del flujo de trabajos. Y los usuarios de

sistemas de conferencia electrónica o de WWW consideran como nudo de groupware al acceso compartido de la información.

Estas diferentes vistas se producen porque groupware tiene sus raíces en tres áreas de aplicación que, si bien son distintas, cada vez más tienden a solaparse. Estas tres áreas son: envío de mensajes electrónicamente, manejo de información y automatización de proceso/flujo de trabajos.

Mirando groupware desde este punto de vista, es claro que no puede definirse como una tecnología simple, ni como una colección de aplicaciones. Los grupos necesitan cambiar todo el tiempo, y cada grupo es distinto de los demás. Por lo tanto, para que un sistema de groupware sea efectivo, debe ser capaz de soportar todos los modos de trabajo en grupo. Específicamente:

- El envío de mensajes electrónicos es una herramienta efectiva para notificaciones y satisface a la mayoría de las necesidades de comunicación.

- Las tablas boletín y los sistemas de conferencia (bases de datos compartidas) se usan para proveer una visión más coherente y común de la interacción del grupo.

- Pero, los sistemas donde la mayoría de la información de los grupos se guarda en bases de datos, sufre de la misma pasividad que la tecnología de bases de datos convencional. Esto es, cada participante de un grupo tiene la responsabilidad de encontrar la información y planear sus acciones de acuerdo a los cambios de esa información. De aquí que surge la necesidad del uso coordinado de las tecnologías de: envíos de mensajes, para notificación, y de: bases de datos, para compartir la información.

- Un sistema de flujo de trabajos basado únicamente en formularios que se rutean de una persona a otra, nuevamente deja a cada persona del grupo sola, sin una visión del proceso completo. Por otro lado, un sistema de flujo de trabajos basado en almacenar la información clave y su estado en bases de datos, puede proveer de un contexto para quien lo necesite, usando siempre envíos de mensajes para notificaciones. Aquí, groupware representa la convergencia de envíos de mensajes, manejo de información y automatización del flujo de trabajos, reuniéndose así las necesidades de los grupos de crear, compartir y beneficiarse del conocimiento colectivo.

5.3.- Esquemas de clasificación

Hay tres esquemas de clasificación útiles para groupware:

1.- El más conocido es el esquema del Dr. Robert Jahansen, el cual se centra en tiempo y espacio o interacciones: mismo tiempo - diferente lugar, mismo tiempo - mismo lugar, diferente tiempo - mismo lugar, diferente tiempo - diferente lugar, cualquier tiempo - cualquier lugar.

2.- El esquema de Esther Dyson se centra en el control: centrado en el usuario (maneja el trabajo localmente), centrado en el trabajo u objeto o centrado en el proceso.

3.- El tercer esquema está orientado al producto y se centra en la funcionalidad del mismo, tal como:

- correo electrónico y mensajes
- conferencias
- sistemas de soporte para decisiones de grupo
- manejo de la documentación de grupos
- workflow (flujo de trabajos)
- utilidades para el trabajo en grupo y herramientas de desarrollo
- ambientes para groupware
- servicios de groupware
- aplicaciones de groupware

5.4.- El mercado

La tecnología de groupware tiene alrededor de 20 años, pero su popularidad actual se debe a la convergencia de tendencias técnicas, económicas, sociales y organizacionales. La infraestructura de redes recién ahora es suficiente como para soportar grandes grupos regionales, nacionales y globales. El costo de las redes y del software de groupware ha bajado. Los empleados desean cada vez más una mayor autonomía. Las compañías desean mayor productividad, aumento de calidad, menos encuentros, mejores servicios para los clientes, automatización de los procesos de rutina, integración de equipos geográficamente separados y nivelación de experiencia. Groupware es la principal herramienta usada para implementar programas de reingeniería.

El mayor segmento de crecimiento del mercado es el del correo electrónico y workflow, pero se espera que workflow crezca aún más y que el correo electrónico decrezca debido a que su funcionalidad se está integrando en otras tecnologías. [18]

5.5.- Tendencias

Debido a la expansión, contracción y maduración de los productos, la categorización de los productos de groupware debería reducirse a cuatro: correo/mensajes electrónico, manejo de documentación de grupos, herramientas para workflow y, herramientas para desarrollo/utilidad del trabajo en grupo.

Para el fin de este siglo casi todas las aplicaciones estarán habilitadas para el trabajo en grupo, para workflow o bien soportarán equipos sobre redes, lo que significa que todo el software será groupware. Groupware no será más un término útil. Ya los sistemas operativos de Microsoft y UNIX incluyen correo electrónico, calendarios y soporte de Internet. [18]

5.6.- Motivaciones

La siguiente lista representa las principales motivaciones para usar groupware:

- Mejor control de costos.
- Incremento en la productividad.
- Mejores servicios a los clientes.
- Menos reuniones.
- Automatización de procesos de rutina.
- Extensión de la organización para incluír tanto al cliente como al proveedor.
- Integración de equipos geográficamente separados.
- Mejor coordinación global.
- Proveer un nuevo servicio que diferencie a la organización.
- Nivelación de la experiencia profesional.

Groupware usa la tecnología para proveer soluciones a los procesos empresariales. Observando más de cerca, vemos las principales siete fuerzas que dan la propulsión inicial hacia groupware:

- Está disponible una infraestructura de red capaz de soportar groupware.
- El software y el hardware de groupware ha mejorado el precio y la perfomance haciéndose, de esta forma, disponible a más cantidad de personas.
- La recesión mundial está forzando a incrementar la productividad.
- Vendedores conocidos como Microsoft, WordPerfect, Lotus, IBM, etc. están promoviendo los productos de groupware, incrementando así el conocimiento sobre ellos en el mercado.
- El incremento en la competencia impone cambios en las organizaciones, haciéndolas más chatas y flexibles, y a menudo, requiriendo groupware para esta transformación.
- La creciente complejidad de los productos de hoy en día y los procedimientos empresariales conducen al uso de equipos ad-hoc soportados por groupware.
- Artículos publicados en impresiones empresariales han incrementado el conocimiento que se tiene de groupware y han despertado la curiosidad de los líderes.

Observando el mercado de groupware, vemos que las leyes de la fisica pueden aplicarse a los mercados y a las tecnologías. El mercado de groupware está conducido por tres fuerzas: una fuerza inicial usada para sobrellevar la inercia, seguida de un momento y finalmente, una reacción que es igual y opuesta a la fuerza inicial.

Hay también fuerzas iguales y opuestas que inhiben el crecimiento de groupware, incluyendo:

- Un bajo nivel de educación sobre groupware en la comunidad empresarial.
- Confusión en el mercado sobre la naturaleza de groupware.
- Debido a la recesión muchas firmas perciben que no pueden afrontar una inversión en groupware.

- Los canales de distribución son nuevos y no están completamente implementados.
- Los negocios MIS se preocupan de que empezarán a depender de los vendedores de groupware.
- Las organizaciones se resisten a los cambios.
- Hay algunos standards en el merdado de groupware que alientan el crecimiento rápido. [17]

5.7.- Desafíos

La mayor característica tecnológica es integrar los sistemas de groupware con sistemas de mando, para compartir datos. Groupware involucra la colaboración o el compartir información, lo cual ha sido tradicionalmente atesorado. Groupware soporta estructuras organizacionales basadas en equipos planos o de proyecto, en lugar de estructuras organizacionales jerárquicas. [18]

5.8.- Comunicación, colaboración y coordinación

[1] Mirando groupware desde una perspectiva más orgánica, queda claro que el conocimiento y la información se comparten en el soporte de tres funciones primarias que son: comunicación, colaboración y coordinación. A su vez, hay dos dimensiones que caracterizan el rol que juega la tecnología para facilitar el trabajo en grupo: el grado de estructura impuesto por la tecnología y el rol pasivo/activo que juega la tecnología como guía del grupo de trabajo.

La primera dimensión trata de la variación del grado de estructura que se necesita para el trabajo en grupo. Este puede variar desde situaciones donde la información está distribuida en forma ad-hoc hasta procesos más estructurados, donde los pasos están predefinidos.

La segunda dimensión se orienta a la relación pasiva/activa entre el medio tecnológico y el individuo o grupo de trabajo. Las aplicaciones pasivas dejan el control en manos del usuario o del grupo, mientras que las activas juegan un rol directivo, controlando el flujo de trabajo del grupo. Por ejemplo, un sistema de bases de datos compartidas que permite a los usuarios navegar a través de una discusión, es pasivo, mientras que un sistema que activamente monitorea un proceso y notifica al usuario de los respectivos eventos, es activo.

Entonces, una definición de groupware desde esta perspectiva es simplemente: "herramientas para permitir que las personas trabajen juntas a través de la comunicación, colaboración y coordinación". De hecho, lo que hace poderosa a una plataforma de groupware es su habilidad para soportar los movimientos dinámicos entre comunicación, colaboración y coordinación.

5.8.1.- Comunicación

5.8.1.1.- Definición

La comunicación es la transmisión de conocimiento. En el ambiente comercial, los colegas se comunican entre sí en distintas formas: reuniones formales y presentaciones, memos interoficinales, por teléfono y charlas informales. La información y el conocimiento impartido en estas interacciones toma la forma verbal (hablado y escrito) y visual (imágenes, gestos, dibujos, lenguaje del cuerpo).

El rol de un sistema de comunicación es el de ser un medio electrónico para transmitir información. Variables como tiempo, lugar y número de participantes permitirán determinar cuál es el sistema de comunicación más apropiado para cada circunstancia.

5.8.1.2.- Envío de mensajes electrónicamente: una tecnología para la comunicación

El envío de mensajes electrónicamente es un transporte de la forma guardar y seguir de objetos electrónicos entre personas, entre personas y aplicaciones y entre aplicaciones. El punto de diseño del envío de mensajes electrónicos es la transmisión asincrónica de mensajes de un lugar a otro. Los mensajes pueden contener información simple o compleja, y pueden ser enviados a individuos específicos o a grupos. El envío de mensajes soporta compartimiento de información en distintos tiempos y lugares, esto es porque la información es apilada en el receptor por el que la envía.

La forma de transporte guardar y seguir es usada para mover o apilar objetos de un punto a otro a través de una gran cantidad de puntos intermedios hasta que llegan a su receptor final. El envío de mensajes provee conectividad asincrónica porque el que envía y el que recibe no necesitan sincronizarse en el tiempo.

El envío de mensajes se lleva el crédito de haber revolucionado la comunicación de uno a muchos. Naturalmente, enseguida apareció la necesidad de la comunicación muchos a muchos. El uso de correo electrónico para este último tipo de comunicación aumentó exponencialmente el volumen de mensajes y, esta transición no es, ni remotamente, un pequeño agregado al envío de mensajes en la forma guardar y seguir.

5.8.1.3.- Almacenamiento de mensajes

Tradicionalmente, el almacenamiento de mensajes ha servido como depósito temporario de los mensajes que serían ruteados a sus destinos. Con el tiempo, como el envío de mensajes fue cada vez más usado para comunicaciones uno a muchos y muchos a muchos, el almacenamiento empezó a ser el contenedor de grandes cantidades de información colectiva.

Debido a que el diseño original se basó en el almacenamiento de mensajes de una forma temporaria, éste tuvo que ser optimizado para que pudiera devolverlos y enviarlos. Este modelo asume que los documentos y objetos que se rutean usando el sistema de

correo electrónico, serán almacenados de alguna forma (por ejemplo en el disco duro de una PC) por la persona que envía y por los receptores. El almacenamiento de mensajes no fue diseñado para el manejo y conservación de la información de manera persistente.

Por todo esto, usuarios y proveedores han analizado formas de manejar, manipular y automatizar el uso de esta información a través de distintas maneras de clasificaciones, procesos basados en reglas y desarrollo de aplicaciones. Aparecieron las API (Application Programming Center) para envío de mensajes, como una forma de usar sistemas de envío de mensajes para mover objetos de un usuario a una aplicación, de una aplicación a un usuario y entre aplicaciones.

Hay dos aspectos que desafían al manejo de la información de las empresas. Primero, los usuarios pueden manejar volúmenes de mensajes cada vez más grandes. Segundo, la falta de herramientas sofisticadas para el manejo de información, que permitan manipular dicha información contenida en el correo electrónico, ha comenzado a ser un problema cada vez mayor.

El correo electrónico es víctima de su propio éxito. Su fácil uso y su amplia adopción ha llevado a una explosión en el tráfico de mensajes. Esto lleva a que los usuarios pierdan información, y a que acumulen referencias a mensajes no contestados. Se pierde tiempo en leer y visar mensajes de poca importancia. Algunos usuarios están tan sobrecargados de mensajes diarios que han empezado a ignorar muchos de ellos como si fueran del montón. Esta situación tiene un impacto negativo en la productividad individual y comercial, produciendo pérdida de información y reduciendo la responsabilidad del cliente.

Además, el problema sólo puede empeorar. La creciente capacidad de los sistemas de correo electrónico (por ejemplo, enviar archivos de 50 megabytes a más de diez usuarios) incrementa el problema. Inicialmente, había un impacto positivo en la productividad porque las personas obtenían, más rápidamente, la información que necesitaban. Sin embargo, al aumentar el volumen y la complejidad de la información, los usuarios llegaron al punto de disminuir la productividad, debido a que la información que se necesita es confundida con la información que los demás piensan que se necesita o se quiere.

Si el margen de recepción de información excede nuestra posibilidad de absorberla y manejarla, llegamos a un punto en que debemos disminuir el retorno. En otras palabras, el correo electrónico soluciona efectivamente el problema de distribución de información para el que envía, pero genera un manejo de información tremendo para su receptor.

Este modelo de comunicación, si bien es efectivo para soportar comunicación no estructurada, falla en el soporte de niveles de interacción más complejos. En contraste, las tecnologías diseñadas para soportar interacciones complejas para colaboración de muchos a muchos usan un modelo totalmente diferente para el manejo y distribución de la información.

5.8.1.4.- Conclusiones

Determinamos que el correo electrónico es un medio efectivo para comunicaciones de uno a uno y uno a muchos. Pero, debido a la falta de estructura y al volumen de información a ser transferido, la comunicación de muchos a muchos se hace inmanejable en este ambiente.

Para determinar la tecnología apropiada para resolver estos problemas, es importante distinguir entre: envío y manejo de la información. Para solucionar el problema del manejo de información, se deben buscar otras tecnologías.

5.8.2.- Colaboración

5.8.2.1.- Introducción

La colaboración se basa en un espacio compartido. Puede ser una habitación, un pizarrón, o un espacio compartido on-line. Dicho espacio sirve como comienzo para el acto de colaborar y es esencial como medio para manejar la ambigüedad propia de la interacción humana.

La colaboración se puede dar entre dos personas, o se puede tratar de compatir información de muchos a muchos. Distintas formas de colaboración pueden ser: resolución de problemas , brainstorming, identificación y localización de datos, etc.

Una de las contribuciones más importante de la tecnología al área de la colaboración fue la eliminación de los requerimientos de tiempo y espacio. Los encuentros cara a cara son posibles en los casos en que los miembros de un grupo pueden compartir el mismo tiempo y el mismo lugar físico. Los teléfonos han eliminado muchas barreras de ubicación. De hecho, la experiencia muestra que una vez que los grupos han incorporado tecnologías de colaboración a su ambiente, se minimiza la cantidad de encuentros cara a cara los cuales, de otra forma, serían necesarios para el intercambio de información e ideas. Todo esto permite a los grupos optimizar los encuentros cara a cara, limitándolos a los casos en que sean necesarios o más provechosos.

5.8.2.2.- Bases de datos compartidas: una tecnología para la colaboración

Debido a su naturaleza asincrónica y a su gran disponibilidad, el envío de mensajes se ha comenzado a usar como un medio para la colaboración, la cual, si bien pareciera ser una extensión lógica de los mensajes, no es tan así, ya que el modelo de envío de mensajes falla en ciertos aspectos.

Los sistemas de envío de mensajes se concentran principalmente en archivos de pistas que guardan mensajes en relación con sus enviadores y receptores. Por lo que, para un usuario, es complicado encontrar la información por temas. Además, mantener un contexto para desarrollar una discusión basándose en los correos electrónicos es totalmente complicado porque no se puede determinar quién respondió a qué y en qué

orden. Todo esto llevó a que se extendieran los sistemas de correo electrónico como para que soporten un espacio de discusión compartido, lo cual introduce cierta estructura como: roles, control de acceso y disponibilidad de una estructura conversacional para el usuario.

Las bases de datos compartidas facilitan la interacción colaborativa proveyendo un espacio de trabajo virtual y común con una interface orientada al grupo, que permite a los participantes compartir información e ideas. En contraste con el envío de mensajes, que usa un modelo de envío o apilado, la tecnología de bases de datos compartidas soporta un modelo de desapilado para compartir información. El modelo de desapilado permite a los usuarios recuperar información según la vayan necesitando. Los usuarios tienen mayor control cuando se unen a varios grupos de discusión. Tampoco se atan más a un modelo de correo electrónico, sino que asumen la responsabilidad de recuperar información (o ignorarla) según su propio criterio.

Las bases de datos compartidas también difieren de los sistemas de envío de mensajes en que no sólo guardan un conjunto de mensajes, sino que también guardan items de discusión, documentos soporte, argumentos, etc. en un lugar común, que puede ser visto a través de una estructura común y además, se registra todo lo ocurrido. Esto facilita la comprensión y es fundamental para permitir que los colaboradores asimilen colectivamente conceptos y resultados claves.

Además, donde se provee una vista compartida, se requieren distintas formas de presentación debido a que cada individuo querrá una vista privada para sus tareas. Estas distintas formas deben incluir que un usuario puede ver la información por fecha, por autor, por tipo de documento, etc. También deberían dar la posibilidad de presentar la misma información, de diferente forma, según el usuario. Por ejemplo, a los de marketing se les debería presentar la información por nombre de cliente o por segmento de mercado, mientras que a los ingenieros se le debería presentar por número de parte o por nombre del producto.

Hay que tener en cuenta la importancia de ciertos atributos como son: la habilidad en adaptar interfaces de usuario, la granularidad con que puede ser vista y manipulada la información, la habilidad para definir variables asociadas al sistema. Los productos comerciales que son una aplicación específica, sin proveer acceso a la plataforma de bases de datos subyacente, están muy limitados. Mientras que, productos que exponen sus plataformas de bases de datos subyacentes, y ofrecen herramientas y aplicaciones adaptables, proveen mucha mayor flexibilidad.

5.8.2.3.- Aplicaciones colaborativas

Al desarrollar los requerimientos para un ambiente de colaboración flexible y adaptable, es importante distinguir entre la tecnología y las aplicaciones implementadas con esa tecnología. Una base de datos compartida y flexible como centro de un sistema de groupware provee la plataforma sobre la cual se puede desarrollar un amplio rango de aplicaciones, que pueden variar desde simples bases de datos de discusión, hasta bases de conocimiento que soporten sistemas de asistencia al cliente y respuestas a preguntas.

El sistema de conferencia Electronic Conferencing Electronic permite la colaboración asincrónica introduciendo cierta estructura que facilita el proceso de compartición, organización y navegación de la información a través de un espacio electrónico interactivo que sirve como depósito de contribuciones.

Los trabajos para resolución de problemas, reúnen tareas de propósito general como brainstorming, para generar ideas; luego permiten la estructuración de esas ideas y por último su evaluación. Los sistemas de conferencia electrónica usan la tecnología de bases de datos compartidas para proveer la estructura necesaria a fin de facilitar dichos pasos. En las conferencias electrónicas, los mensajes se guardan en una base compartida y todos los participantes pueden ver los nuevos, y responderlos. Puede haber un moderador que comience a tratar nuevos temas para llevar así al grupo de un estado del proceso a otro.

5.8.2.4.- Bases de datos compartidas en otras aplicaciones

Una base de mensajes estructurada se puede usar para proveer un mejor entendimiento de la discusión. Por ejemplo, en el caso de un grupo de autores, todos deberían compartir un esquema común del documento que están escribiendo. Ese documento puede almacenarse como una colección de párrafos, capítulos y secciones. Estos pueden ser vistos: linealmente, como en cualquier procesador de textos; en modo revisión; nuevas y viejas versiones; etc. Además, cada párrafo puede servir como tema de discusión y las conversaciones sobre el mismo pueden estar estructuradas en relación a un hecho o problema en particular, mientras que en una conferencia on-line común sólo se tendrían notas sueltas sobre los distintos temas.

Las aplicaciones que permiten trabajar en grupos, como ser los procesadores de texto, etc. que guardan su información en bases de datos (no en archivos no estructurados), son mucho más poderosas. Una vez que la información fue capturada en una base de datos común, ésta puede verse a través de la aplicación o a través de interfaces de groupware genéricas que faciliten la discusión, el debate y la toma de decisiones.

5.8.2.5.- Sistemas de publicación de referencias

Los sistemas de publicación de referencias se consideran groupware porque facilitan el compartimiento de información.

Para entender mejor sus roles en el ambiente empresarial, es útil comparar y contrastar las publicaciones de referencias con aplicaciones colaborativas. Ambas tienen una estructura similar, pero las aplicaciones colaborativas son interactivas mientras que las de publicación de referencias son unidireccionales.

En realidad, las publicaciones de referencias son típicamente unidireccionales, por lo tanto es debatible si pueden ser llamadas aplicaciones colaborativas o aplicaciones de comunicación.

Un sistema de publicación de referencias es una base de datos distribuida que contiene documentos terminados. Si la tecnología subyacente ofrece, adicionalmente,

facilidades para la colaboración y la comunicación, entonces este sistema puede transformarse en un ambiente dinámico, que podrá soportar el aprendizaje y la construcción de visiones comunes, que es mucho más que un ambiente de recuperación y búsqueda de información. Por ejemplo, un equipo podría usar una base de datos de discusión para colaborar en el alcance, dirección y detalles de una estrategia para un producto. El resultado de esta colaboración será un documento estrategia, el cual se almacenará en el sistema de publicación de referencias correspondiente. Los lectores podrán, entonces, armar respuestas, recomendaciones y predicciones al respecto. Luego, la compañía podría incluír un link entre el documento estrategia y la base de datos de discusiones, a la cual todos los lectores pueden contribuir y responder. Por otra parte, los contribuyentes a la base de datos de discusiones pueden incluír punteros a documentos en otros sistemas de publicación de referencias como parte de sus comentarios.

5.8.2.6.- La naturaleza pasiva de las bases de datos compartidas

El modelo de bases de datos compartidas para el almacenado y mantenimiento de información on-line brinda muchas ventajas sobre el modelo basado en envío de mensajes. En principio, la información es recuperada por el consumidor cuando éste la precisa, permitiendo además un mayor control sobre la información recibida y consumida. Sin embargo, cuando se usa esta tecnología sin notificaciones o soporte de mensajes, también presenta limitaciones. Como las bases de datos compartidas dependen de que el consumidor busque la información, éstas se transforman en bases pasivas. Entonces, es importante que se les agregue la facilidad de notificar a los usuarios del agregado o modificación de información.

De la misma forma en que el correo electrónico llegó al punto de sobrecarga de información, una explosión de docenas, cientos o miles de bases de datos compartidas dejarían al usuario en un ambiente superpoblado y confuso. Un fenómeno similar ocurre en WWW con la aparición de miles de páginas home cada mes.

5.8.2.7.- WWW como una herramienta de colaboración

Con el desarrollo de WWW existe la oportunidad de compartir y comunicar información desde un grupo a un nivel global. Los mecanismos para lograrlo son servers WWW y programas clientes, además de un conjunto de propósitos standard que les permitan comunicarse para identificar objetos de información y para estructurar y ver la información, tal que ésta pueda ser pasada vía links hipertexto. [9]

El WWW es un conjunto de protocolos que operan sobre Internet. Estos protocolos sirven como base para un ambiente cliente/servidor que soporta compartimiento de información, procesamiento de transacciones y comercio electrónico.

La infraestructura del WWW consta de tres tecnologías esenciales que definen la comunicación entre un cliente y un server, conectados sobre una red TCP/IP. Estas tecnologías son:

- *HTTP*: HyperText Transport Protocol, gobierna la comunicación entre el cliente y el server de la red.

- *HTML*: HyperText Markup Language, es el formato de documentos para las páginas de la red. HTML usa directivas para especificar el formato de las páginas, dejando que el mismo lo defina el cliente a partir de ellas. HTML es un subconjunto de SGML (Standard Generalized Markup Language) que provee códigos que se usan para darle formato a los links entre documentos, dando lugar así a un hipertexto.

- *URL*: Uniform Resource Locator, es un puntero a un recurso Internet. Sirve como un esquema de direccionamiento general para las páginas guardadas en el server del WWW. El URL contiene el nombre del server y las posibles extensiones que definen una página de WWW específica, u otra información a ser usada por el server.

Mientras que muchas personas ven a WWW como centralizado en productos, no hay duda de que esto cambiará rápidamente y que lo veremos como un conjunto de capacidades implementadas con protocolos específicos que sean soportados por un gran rango de productos.

Con el tiempo, los protocolos WWW serán los que determinen y limiten lo que se pueda hacer o no en WWW. La evolución de los protocolos (específicamente HTTP y HTML) han madurado en dos fases, y están entrando en una tercera:

- *Fase uno - Publicación de información*: La primera versión de HTML soportaba sólo publicación de información con un modelo de hipertexto. Cualquiera podía crear páginas HTML y almacenarlas en un server HTTP, y cualquiera con un browser podía usar HTTP para acceder a las páginas de WWW. Rápidamente, los desarrolladores de WWW, notaron que había tareas que no podían realizar con protocolos simples, por lo que aparecieron las aplicaciones de salida o "trap door", que fueron la interface de salida común (CGI). Un URL podía también referenciar a un programa que sería invocado en el server WWW a través de CGI. Esto abrió un nuevo mundo para WWW, pues podía soportar publicaciones dinámicas. Ahora, la mayoría de la interacción de WWW es a través de páginas creadas dinámicamente, vía programas CGI. Entonces, si bien el factor de enlace para la construcción de este tipo de aplicaciones no es HTTP y HTML, sí es la clase de ambiente de desarrollo de aplicaciones que se dispone para escribir programas CGI, que interactúen con fuentes de datos externas. Por esto, está surgiendo una nueva generación de herramientas de desarrollo de aplicaciones para el WWW. Es razonable que los ambientes de desarrollo de aplicaciones que ya soportan aplicaciones colaborativas, también soporten protocolos WWW.

- *Fase dos - Llenado de formularios*: Se trata de consultas interactivas. Para soportarlas, se debe poder presentar al usuario una página para, por ejemplo, poder ingresar una palabra específica que se quiera encontrar. La extensión de HTML para soportar esto, fue el mayor paso en la evolución de WWW. Una vez que los datos fueron ingresados en un campo, éste invoca al programa CGI, vía un URL al server, que extrae los datos del formulario, procesa la información, crea dinámicamente una página y la envía de regreso al usuario. El llenado de formularios no sólo permite a WWW soportar aplicaciones de

consultas interactivas, sino que también permite el comercio electrónico y los servicios de transacciones (por ejemplo, sistemas de reservación de pasajes aéreos, servicios de manejo de stock, catálogos, etc.). Las ventajas del procesamiento de transacciones basadas en WWW para el proveedor son:

- conectividad por todo el mundo a través de Internet
- soporte universal para cualquier máquina cliente que tenga un browser de WWW

- *Fase tres - Lenguajes de programación*: La versión original de HTML especificaba cómo debería ser el formato de los documentos. Mirando esto desde otro punto de vista, HTML, era realmento un lenguaje de programación simple, y es la evolución de HTML la que define esta tercera fase. De la misma manera que los lenguajes de programación, HTML también evoluciona hacia un modelo orientado a objetos. Los browsers pueden incluír código que puede ser interpretado por un programa y las páginas de WWW pueden incluír programas. Cuando se accede a una página y se la muestra mediante un browser, éste interpreta el programa y ejecuta la acción requerida, por ejemplo, proveer de una animación a una imagen.

5.8.2.8.- Conclusiones

Al examinar la mezcla de tecnologías basadas en comunicación y colaboración notamos que:

- Los requerimientos de colaboración y comunicación son distintos. Además, el envío de mensajes electrónicos solo, no facilita lo suficiente el proceso de colaboración. La tecnología de base de datos emplea un modelo de desapilado de información de distribución, que ayuda a los usuarios en el proceso colaborativo.

- La colaboración requiere de un sistema que combina estos modelos de apilado y desapilado, y provee un ambiente robusto para la explotación de las diferentes formas que necesitan los usuarios para comunicarse y colaborar.

- Una base de datos compartida es esencial para el trabajo en común, para vistas compartidas y para la cristalización de información dentro del conocimiento organizacional. Una forma de mejorar la integración de los modelos de apilado y desapilado, es a través de herramientas que soportan un uso coordinado de tecnologías de envío de mensajes y bases de datos compartidas.

La tercera categoría de trabajo de grupo (coordinación) es soportada por ambas tecnologías, y por herramientas que permiten a los grupos programar su uso combinado.

5.8.3.- Coordinación

5.8.3.1.- Introducción

Hemos discutido cómo hacen los grupos de personas para comunicarse y colaborar y así, compartir información y nivelar conocimiento de manera tal que los ayude a realizar su trabajo en forma más eficiente y efectiva. Lo que es muy característico de esta interacción es su naturaleza no estructurada. Las personas se envían mensajes por correo electrónico a su propia discresión, y se refieren a recursos compartidos cuando lo necesitan. Las actividades se producen dinámicamente según las necesidades.

Sin embargo, muchas actividades comerciales son mucho más estructuradas por naturaleza. Las empresas no esperan que las personas colaboren para procesar un reporte muy costoso. La política de las empresas debe especificar la coordinación que se requiere entre estas personas para lograr un objetivo determinado. El éxito en la culminación de un proceso de negocios predefinido depende de la coordinación de las personas, en cuanto a la completación de un conjunto de tareas estructuradas en una secuencia particular y en un tiempo esperado. Mientras que la colaboración es relativamente pasiva desde la perspectiva de un sistema (creamos un espacio de trabajo común pero no determinamos cómo debe ser usado ese espacio), la coordinación es muy activa desde la perspectiva de un sistema (especificamos cómo se deben acompañar las actividades). Cuando elegimos movernos de la colaboración a la coordinación para un problema específico, implementamos sistemas de flujo de trabajos en los cuales definimos formularios, especificamos operaciones y un ruteo lógico para ellos, detallamos cómo se deben considerar los datos externos, etc. La coordinación se refiere al uso de herramientas de desarrollo de aplicaciones para una clase de ellas, a la que nos referimos generalmente como flujo de trabajos y donde el principal atributo es el seguimiento de algunos recursos. La herramienta esencial para la construcción de sistemas de coordinación es un ambiente de desarrollo de aplicaciones.

La mayoría del trabajo involucra una combinación de procesos y tareas altamente estructurados, donde se complica el proceso, mientras que, las reglas, rutas y roles son definidos dinámicamente, a medida que se hace el trabajo. Esto es porque los sistemas de flujo de trabajos solos, sin componentes colaborativos y de comunicación que provean la interacción de software, no tienen éxito y pecan de demasiado rígidos.

La coordinación, entonces, es más que la automatización de una secuencia de tareas estructuradas haciendo que las personas entren y salgan del proceso según las necesidades. Vemos que el conocimiento que se necesita para terminar una tarea, se adquiere como resultado de las relaciones entre los participantes, fuera del contexto del proceso mismo. La coordinación completa incluye soporte para conversaciones individuales que permitan a las personas mejorar la información que necesitan para hacer su trabajo, especialmente cuando estas conversaciones ocurren en el contexto de un proceso más estructurado.

Una solución integrada que coordina el uso de notificación de mensajes y bases de datos compartidas para la colaboración, provee una aproximación balanceada para soportar procesos estructurados y no estructurados.

5.8.3.2.- Cuatro generaciones de envío de mensajes

Las cuatro generaciones de sistemas de mensajes explotan el uso coordinado de mensajes con bases de datos compartidas. Si miramos hacia atrás para ver cómo han evolucionado los sistemas de mensajes, notamos que los de la primera generación eran sólo capaces de soportar mensajes de texto simples. Los de la segunda generación agregaron la capacidad de asociar documentos binarios a mensajes de texto simples. La tercera generación de sistemas proveyó soporte para textos ricos, por ejemplo, color, múltiples fuentes de caracteres, tamaño de caracteres, etc. y embebió objetos en el cuerpo del mensaje mismo y, finalmente, la cuarta generación, soporta hipertexto con links hacia documentos pertenecientes a bases de datos compartidas y a sistemas de archivos. Más que asociar un objeto a un mensaje, incluímos en el mensaje un "doclink" que es un puntero electrónico al objeto de la base compartida. Cuando un usuario hace doble click sobre el "doclink", el objeto es devuelto y presentado al usuario automática e inmediatamente. Los sistemas de mensajes mejoraron ya que no se precisa más que se asocien objetos a los mensajes. Las bases de datos mejoraron porque los usuarios ahora tienen un medio para notificar a otros de la existencia de información importante o relevante, que de la otra forma languidecería en la base de datos, sin ser usada ni notada. Todo esto es lo que llamamos sistemas de mensajes y groupware.

5.8.3.3.- Integración de mensajes y groupware

Los usuarios de correo electrónico piensan en groupware en términos de mensajes, mientras que los usuarios de conferencias y de sistemas de publicidad on-line ven a groupware como a una función de las bases de datos compartidas, y los usuarios centrados en la automatización de negocios estructurados se inclinan a percibir a groupware como un flujo de trabajo automatizado. En realidad, cada una de estas tecnologías representa una dimensión específica de groupware. Los mensajes y las bases de datos compartidas se convirtieron en el fundamento de los sistemas de automatización del flujo de trabajo. Los mensajes se trasladaron al espacio del flujo de trabajo, exponiendo sus APIs al desarrollo de aplicaciones y herramientas para crear una técnica de enrutamiento del flujo de trabajo en un proceso automático. Las bases de datos compartidas se han extendido para soportar el seguimiento de la automatización del flujo de trabajos.

5.8.3.4.- Modelo de mensajes del flujo de trabajos

La automatización del flujo de trabajos está asociada con el enrutamiento automático de documentos. El flujo de trabajos automático basado en rutas usa, generalmente, el sistema de mensajes subyacente para enrutar a los documentos hacia la próxima persona que debe realizar una acción. La ruta puede codificarse por hardware, o puede ser una regla que determine el camino basándose en un valor específico o en el rol de una persona. Estas reglas pueden ser sofisticadas y aún llamar a una aplicación externa que devuelva ciertos datos.

Hay una desventaja significativa: como el documento está en ruta, ninguna otra persona que no sea la que lo tiene en su correo, puede acceder a este documento. Vemos el problema con un ejemplo de la negociación de un contrato: para firmar el

contrato, éste debe ser aprobado internamente por varias personas. Un sistema de flujo de trabajos basado en correo electrónico enruta el contrato a cada una de las personas de las cuales se requiere la aprobación, negación y/o comentarios. Supongamos que el cliente quisiera conocer el estado del contrato, o que quisiera hacerle cambios durante el proceso de aprobación. ¿Es posible determinar quién lo tiene?. Si es así, ¿es posible devolverlo?, ¿en qué punto pueden introducirse cambios?, ¿qué ocurre con el proceso de aprobación una vez que se ha implementado un cambio?.

Es posible anticiparse a algunos problemas reconstruyendo ciertas reglas o agregando nuevas reglas, pero la construcción de aplicaciones estratégicas como un parche no es bueno para las compañías y, preveer todas las condiciones y excepciones es imposible.

5.8.3.5.- Modelo de bases de datos compartidas para flujo de trabajos

El segundo modelo de flujo de trabajos es el de bases de datos compartidas. En este modelo, los usuarios consultan una base de datos de pistas para chequear documentos específicos.

El modelo de bases de datos compartidas tiene tres ventajas. Primero, la base está situada en el server y está sujeta a procesos basados en él, que pueden iniciar una acción sin una actividad específica de un usuario. En muchos casos, la acción puede ser un resultado directo de la falta de actividad de los usuarios o una condición externa.

Segundo, el modelo de bases de datos compartidas mantiene el documento o lo registra como disponible para los demás mientras se produce el flujo de trabajos. En el ejemplo de la aprobación del contrato, los cambios se pueden hacer sobre el documento original que está en la base de datos, y el flujo de trabajos puede continuar o ser abortado y recomenzado.

La tercera ventaja es que el modelo de bases de datos compartidas hace que el manejo y el macro-manejo del flujo de trabajos sea más fácil. El server puede monitorear instancias específicas del proceso y mantener estadísticas sobre él, permitiendo mejor manejo y planificación del flujo de trabajos.

La principal desventaja es la falta de notificaciones basadas en eventos por parte del sistema hacia los participantes del flujo de trabajos. Esto significa que el chequeo de la base de datos es incumbencia del usuario.

5.8.3.6.- Un modelo integrado

Ya hemos visto las ventajas de un sistema de bases de datos compartidas y mensajes totalmente integrados a través de cuatro generaciones de sistemas de mensajes. Si generalizamos este modelo, es obvio que las aplicaciones de flujo de trabajos construidas sobre plataformas comunes, que nativamente soporten ambos modelos (un subsistema de mensajes con capacidades de enrutamiento condicionales y bases de datos compartidas para el almacenamiento, devolución, vista y manejo de los trabajos en proceso) provee la robustez y la flexibilidad necesaria para automatizar efectivamente el proceso de trabajos.

Para ilustrar, veamos cómo podría implementarse el mismo ejemplo del contrato usando un modelo integrado: el contrato se crea y se guarda en la base de datos. Cuando es salvado, se envía un mensaje de correo electrónico al primer aprobador. El mensaje no es el contrato en sí mismo, sino que se le dice al aprobador dónde encontrarlo. El mensaje contiene un link de hipertexto al contrato, el cual, cuando es activado, levantará el contrato de la base de datos para que lo use el aprobador, mientras se lo mantiene igual en la base. Se siguen haciendo nuevas aprobaciones, pero la última versión del contrato siempre se mantiene disponible en la base de datos. Cualquier persona que tenga acceso a la base de datos, puede hacer cualquier consulta sobre el estado del mismo.

5.8.3.7.- Modelo de transacciones extendido

Como hemos visto, hay dos aproximaciones básicas a los sistemas de flujo de trabajos, una basado en el modelo de mensajes y otra basado en el modelo de bases de datos compartidas. Los sistemas de flujo de trabajos que soportan ambos modelos, y soportan funciones colaborativas menos estructuradas, serán más exitosos.

Los sistemas de flujo de trabajos raramente existen aislados. Generalmente, los procesos de flujo de trabajos están al frente de la oficina, unidos a sistemas que pasan más desapercibidos y que fueron automatizados mucho antes. Entonces, la unión entre la salida del flujo de trabajos y la entrada de un sistema de procesamiento de transacciones clásico es crítica. Un ejemplo es un sistema de compra/venta simple.

Este ejemplo empieza con una orden de pedido y produce una orden de compra oficial. La entrada al sistema es una orden de pedido aprobada. Sin embargo, una orden de compra aprobada es el resultado de un proceso de flujo de trabajos que comienza cuando alguien decide que necesita comprar algo. Este individuo crea una orden de pedido, la cual, es enviada por el sistema hasta que finalmente sea aprobada o rechazada. Cuando es aprobada, se inicia una transacción en el sentido clásico. En realidad, sin embargo, la transacción comercial comenzó cuando alguien creó la orden de pedido.

Nos referimos a esta visión más comercial de una transacción como un modelo de transacción extendido. Uniendo el flujo de trabajos visible de la oficina con el que está por debajo, se pueden lograr ganancias económicas significativas. Para lograrlo, se necesitan explotar herramientas que permitan interactuar con el sistema de transacciones existente en forma controlada.

5.8.3.8.- Entorno para el desarrollo de aplicaciones

Existe un número de componentes esenciales en el ambiente de desarrollo de aplicaciones para la construcción de aplicaciones de coordinación:

- Se requieren herramientas de diseño y llenado. Los formularios deben incluír texto, imágenes, sonido, campos, botones, listas, etc. y pueden formarse subformularios. Estos subformularios pueden ser ventanas o cajas de diálogo con listas, combo boxes, etc. Se requieren capacidades de edición más potentes para los campos de los formularios, y esta edición puede requerir contactarse con bases de datos externas para validar entradas de los usuarios.

- Se requieren algunas capacidades de programación, en la forma de un lenguaje de scripting, macro lenguaje o un lenguaje de programación completo. Se requiere de un conjunto de APIs para la tecnología subyacente. Los desarrolladores deben ser capaces de proveer instrucciones siempre que ocurran eventos sobre la base de datos como: abrir un formulario, cambiar un campo, agregar un nuevo documento, etc.

- Se requiere una capacidad de agente para acciones que deben realizarse cuando ocurra determinado evento (por ejemplo correr un script determinado).

- Se requiere un ambiente de desarrollo que soporte capacidades ricas de debugging.

- Se requieren vistas definidas por el usuario final para desarrollar aplicaciones de flujo de trabajos, así el resultado serán aplicaciones robustas que exhiban interfaces naturales para el usuario y al mismo tiempo, que tengan el control necesario para aplicaciones comerciales.

Capítulo 6

Trabajo Colaborativo Soportado por Computadoras

La rápida evolución de la información y los nuevos potenciales de la comunicación entre las personas ha sido de gran importancia para el éxito de la mayoría de las organizaciones. Los aspectos claves fueron la creciente disponibilidad de las redes de computadoras y la tendencia hacia el trabajo en grupo. Las actividades en el dominio del trabajo en equipo soportado por computadoras se conocen por las nociones de groupware y de trabajo cooperativo soportado por computadoras. [15]

6.1.- Definición

CSCW es un área interdisciplinaria cuya principal característica es integrar distintas tecnologías para soportar trabajos colaborativos. Combina el entendimiento de la forma en que las personas trabajan en grupo con las tecnologías de redes de computadoras, todo esto asociado con hardware, software, servicios y técnicas.

Las aplicaciones CSCW incluyen el uso de correo electrónico, bases de datos compartidas e hipertextos que abarcan información de la actividad de otros usuarios, videoconferencia, sistemas de chat y aplicaciones compartidas en tiempo real, tales como escritura o dibujo colaborativo.

Las características claves de CSCW son:

- el conocimiento del grupo
- las interfaces multiusuario
- el control de concurrencia
- la comunicación y coordinación dentro del grupo
- los espacios de información compartida
- el soporte de ambientes abiertos y heterogéneos que integran a las aplicaciones monousuarias existentes

Los sistemas CSCW son categorizados de acuerdo a una matriz de tiempo/lugar que distingue entre:

- al mismo tiempo (sincrónico)
- a tiempos diferentes (asincrónico)
- en el mismo lugar (cara a cara)
- en diferentes lugares (distribuido)

6.2.- Un poco de historia

Diez años atrás, Paul Cashman e Irene Grief organizaron un taller de personas de varias disciplinas que tenían un interés común: ver cómo trabajaban las personas, enfatizando en la forma en que la tecnología podría soportarlo. Ellos crearon el término *trabajo cooperativo soportado por computadoras* para describir este interés común. En

la década pasada, miles de investigadores y desarrolladores se interesaron en esto. Las primeras conferencias se realizaron en los Estados Unidos, pero el tema fue llevado enseguida a Europa y Asia, donde ya existía un serio interés y algunos trabajos relacionados.

A mediados de los '60, tareas como completar los asientos de un vuelo de avión o imprimir cheques de pago, se habían transformado en requerimientos que sólo se podían satisfacer con sistemas de mainframe. A mediados de los '70, las microcomputadoras prometieron soporte para grupos y organizaciones, en formas más sofisticadas e interactivas: nació la automatización de oficinas. Las aplicaciones monousuario tales como procesadores de texto, planillas de cálculo, etc. eran un éxito; la automatización de oficinas trató de integrar y extender estos éxitos para soportar grupos y departamentos.

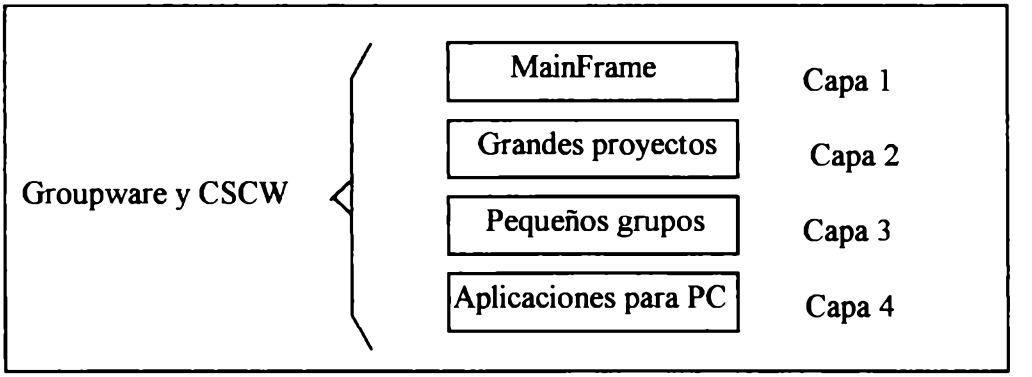
Pero la tecnología no era suficiente. Se necesitaba aprender más sobre cómo trabajaban las personas en grupos y organizaciones y cómo les afectaba la tecnología. CSCW comenzó como un esfuerzo por parte de los tecnologistas de aprender de los economistas, psicólogos sociales, antropólogos, teoristas de organizaciones, educadores y de cualquier otro que pudiera arrojar luz sobre las actividades grupales.

Las aplicaciones incluyen sistemas de videoconferencia, aplicaciones de origen colaborativo, correo electrónico y reuniones electrónicas o sistemas de soporte de grupos. Otros dominios de aplicación que están relacionados, pero no fuertemente representados, incluyen: diseño asistido por computadoras, ingeniería de software asistida por computadoras, ingeniería concurrente, manejo de flujo de trabajos, aprendizaje a distancia, telemedicina y conferencias por medio de redes en tiempo real.

El acrónimo CSCW ha sido criticado porque el trabajo cooperativo es, muchas veces, más bien un objetivo que una realidad. En cambio, *la computación para trabajo grupal* desplaza el enfoque del trabajo soportado por computadoras y lo restringe a pequeñas unidades organizacionales. [16]

6.3.- Contextos de investigación y desarrollo: historia y contribuciones

Existen cuatro capas que representan el enfoque que se tiene con respecto al desarrollo de sistemas de computación y al usuario de la tecnología resultante.



La capa exterior representa los principales sistemas y aplicaciones, sobre todo sistemas de mainframe y grandes minicomputadoras diseñados para satisfacer los objetivos organizacionales: procesamiento de transacciones, control de orden e inventario, etc. La capa más interna representa a las aplicaciones diseñadas para usuarios individuales de PC y estaciones de trabajo: procesadores de texto, debuggers, planillas de cálculo, juegos, etc. Las dos capas intermedias representan grandes proyectos y pequeños grupos respectivamente. El soporte para grandes proyectos incluye reuniones electrónicas y sistemas de automatización del flujo de trabajos, los cuales son útiles para grupos de 6 ó más personas. Por el contrario, el principal enfoque del soporte de grupos pequeños incluye conferencias de escritorio y aplicaciones de escritura colaborativa, las cuales no trabajarán bien para más de 3 ó 4 personas.

Volviendo a la capa más externa, la literatura asociada con sistemas para organizaciones llegó a mediados de los '60, con circuitos integrados y sistemas de computación de tercera generación. Estos se llamaron procesamiento de datos (DP), sistemas de manejo de información (MIS), sistemas de información (IS), y tecnología de información (IT).

Con respecto a la capa siguiente se observa que, a principios y mediados de los '70, la ingeniería de software y la automatización de oficinas, se enfocaron hacia el soporte de computadoras para grandes grupos y proyectos. La automatización de oficinas no sobrevivió, pero muchas de las características subyacentes a los sistemas de manejo de proyecto son consideradas flujo de trabajos.

Lo más reciente es CSCW, que abarca las dos capas centrales y la más interna, en distinto grado, aunque está más ligado a la que soporta pequeños grupos. Por lo tanto, CSCW no está solamente restringido a una capa. Representa una mezcla de características, aproximaciones y lenguajes. CSCW y groupware son difíciles de definir, ninguna definición satisface a los que están involucrados. Se pueden estudiar algunos efectos secundarios como: el uso de aplicaciones en ambientes grupales y organizacionales que fueron desarrolladas para usuarios individuales, las formas en que el software desarrollado para soportar grupos afecta a los individuos y es adaptado a diferentes contextos organizacionales, sistemas desarrollados para soportar objetivos organizacionales pues estos actúan sobre individuos, grupos y proyectos. [16]

6.4.- Cooperación, coordinación y control en el trabajo con CSCW

Los procesos sociales fundamentales y a veces sutiles en el trabajo, tienen una fuerte influencia en la forma en que son adoptadas y usadas las aplicaciones CSCW, además de la forma en que influencian al trabajo subsecuente. Un grupo de investigadores estudia la adopción y uso de aplicaciones CSCW en vivo, es decir en lugares de trabajo que no sean laboratorios.

Muchos artículos sobre CSCW impiden el entendimiento de su buen uso e impacto, ya que se basan en fuertes connotaciones positivas como cooperación, colaboración e imágenes de posibilidades de convivencia, para caracterizar las relaciones de los lugares de trabajo. Por otra parte, tratan de entender los niveles de conflicto,

control y coersión, también comunes en lugares de trabajo profesionales. Virtualmente todos los trabajos tienen elementos cooperativos.

Algunos conflictos de grupo son críticos en cuanto a la identificación de líneas de acción alternativas, para evitar así la conformidad de producción por cantidades, de manera que los conflictos se resuelvan constructivamente y con dignidad. En la práctica, muchas relaciones de trabajo pueden ser multivalentes y pueden mezclar elementos de cooperación, conflicto, convivencia, competición, colaboración, precaución, control, coersión, coordinación y combate. También llaman la atención de tareas importantes como manejo de la organización del trabajo, sociabilidad genuina y aún recreación.

Es común que los profesionales sean competitivos, aún cuando participen rutinariamente en colaboraciones y cooperaciones importantes. Muchos diseñadores de CSCW desean mejorar los elementos cooperativos de la vida laboral. Cuando estos diseñadores evitan las discusiones de control y conflicto, quiere decir que son fundamentalmente ilegítimas y el reconocimiento de su importancia podría aumentar su poder. Desafortunadamente, la práctica común de entender la importancia de los conflictos y el control en el trabajo, hacen que las teorías implícitas, a las que hacen referencia muchos artículos de CSCW, estén limitadas a proveer un entendimiento sobre el uso actual de groupware. La coordinación, por ejemplo, a menudo requiere de un control significativo, pero hay cantidades de coordinación en la comunidad de CSCW que virtualmente ignoran el control como un elemento central.

Los problemas de basarse en conceptos de convivencia para caracterizar las relaciones laborales son más visibles cuando hay contradicciones evidentes entre concepto y ejemplos. Por ejemplo, Ishii y Miyake caracterizan groupware como sistemas que soportan colaboración dinámica en un grupo de trabajo. La actividad sobre la cual demostraron su "Team Workstation" es una tarea instructiva, en la cual una persona entrenada enseña a otra cómo realizar una tarea específica. Ellos notaron, que uno de sus dos instructores se limitaba a indicar procedimientos. Su ejemplo clave nos habla sobre las posibilidades del control autoritario como de la colaboración dinámica.

Los investigadores muchas veces ilustran sus sistemas CSCW con grupos hipotéticos o grupos tomados de universidades o laboratorios industriales, donde el dar y el recibir es algo común. En consecuencia, ellos tienden a no observar si se reflejan las diferencias de status, los sistemas de control organizacional y la jerarquía, en sus prototipos de groupware.

Los desarrolladores de CSCW pueden dar cuentas más realistas de los usos de estas nuevas tecnologías, ya que ellos examinan relaciones sociales multivalentes en los lugares de trabajo: simultáneamente cooperativo, conflictivo, colaborativo, controlado, de convivencia, competitivo. Ellis, Gibbs y Rein caracterizan el problema de la coordinación como la integración y el ajuste armonioso de esfuerzos de trabajo individuales, para acompañar a objetivos más grandes.

Cuando los científicos de la computación hayan estudiado el comportamiento clave que inhibe el rendimiento de un grupo en los lugares de trabajo reales, habrán encontrado un fenómeno interesante. Por ejemplo, Krasher, Curtis e Iscoe estudiaron las

prácticas principales que impidieron el desarrollo de proyectos de gran escala y encontraron una variedad de problemas de comunicación.

Compartir información puede llevarnos a muchas consecuencias distintas, dependiendo de las relaciones sociales entre aquellos que proveen la información sobre sus prácticas de trabajo y aquellos que la usan. [31]

6.5.- Contradicciones de las revoluciones tecnológicas y las transformaciones sociales

Los investigadores de CSCW exploran su caracter especial y a veces lo diferencian de otras tecnologías relacionadas, como automatización de oficinas y sistemas de información. Sin embargo, están comenzando a descubrir aspectos pertinentes del comportamiento organizacional, tales como los dilemas de alterar los patrones de trabajo institucionales y las políticas de los lugares de trabajo.

Desafortunadamente, las personas que están explorando una nueva familia tecnológica: CSCW, manufacturación integrada por computadora (CIM) y sistemas expertos, a veces escriben como si sus nuevas tecnologías pudieran transformar las organizaciones en formas que serían imposibles con las tecnologías precursoras. Suelen adoptar un modo de análisis, utopismo tecnológico, que amplía las posibilidades de valiosos cambios sociales y elimina las posibilidades de pequeños cambios o problemas importantes, considerándolas como aspectos secundarios que vienen con la tecnología. Estas asumciones utópicas hacen difícil de entender a los investigadores, gerentes y profesionales, las oportunidades sociales reales y las limitaciones de una nueva familia de tecnologías.

La computación sola raramente transforma a las organizaciones. Las formas en que los sistemas de CSCW reestructuran las relaciones sociales en el trabajo, dependen de los patrones preexistentes de autoridad, obligación y cooperación, y de una apertura de la organización hacia los cambios, de la misma forma que las capacidades de procesamiento de información dependen de la tecnología. Ver al trabajo y a la organización como personas que trabajan juntas en un sistema con tareas cooperativas, no es lo suficientemente rico como para entender el uso actual y los impactos de CSCW. Las organizaciones deben tener una base más sistemática para entender cuándo van a cumplirse las promesas más excitantes.

Los artículos sobre estos temas ayudan a ilustrar la clase de ideas que ofrece el realismo social a los investigadores interesados en el diseño, uso e impactos de las tecnologías emergentes. [31]

Capítulo 7

AMCO: Una Herramienta para Colaborar - Visión Externa

7.1.- Introducción

Hasta aquí hemos desarrollado, en detalle, el tema colaboración. Ahora, nuestro objetivo es mostrar una aplicación que utilice esta teoría. A tal fin, construimos un ambiente colaborativo (AMCO) que utiliza: como plataforma a Internet, e integra en un solo entorno, herramientas tendientes a facilitar la comunicación, colaboración y por ende, la toma de decisiones de personas, que si bien están en lugares geográficamente distantes, se agruparon con el propósito de realizar una tarea en conjunto. El tema de esta demostración es la toma de decisiones en grupo, teniendo como base, el manejo de páginas WWW. Pero esto no es más que un ejemplo, pues lo que nosotras realmente queremos destacar son las características colaborativas y de interacción humana que son posibles de lograr gracias al avance de la tecnología.

7.2.- Generalidades

AMCO es una demostración de un AMbiente COlaborativo básico. Su desarrollo se apoya en la toma de decisiones en grupo, usando como tema base el manejo de páginas WWW. Dicho tema fue elegido a modo de ejemplo teniendo en mente, tanto la necesidad que tienen muchas personas de expresar sus ideas en Internet, debido a su gran auge, como la posibilidad de construir dichas páginas en forma más simple, gracias a la participación de otros individuos interesados en el mismo tema.

El ambiente AMCO está compuesto básicamente de:

- Un Manejador de Páginas, el cual es una herramienta para tomar decisiones en grupo, usando como tema base el manejo de páginas HTML..
- Un Chat, el cual posibilita la comunicación entre los usuarios.
- Un Visor, para poder observar el estado en que se encuentra cada usuario.

7.3.- Especificación

AMCO tiene como objetivo fundamental integrar en un único entorno herramientas que permitan la colaboración entre personas dispersas por el mundo, con la finalidad de que ellas puedan tomar decisiones grupales con respecto al manejo de páginas WWW, como ya lo explicamos en la sección previa. A continuación detallamos cada una de ellas.

7.3.1.- Manejador de Páginas

Consideramos que las personas interesadas en construír páginas para WWW colaborativamente, tendrán predilección por distintos temas. Por ello, se provee la posibilidad de proponer temas para su posterior desarrollo, quedando éstos a disposición del resto de los usuarios del sistema. Cada uno de ellos puede, o bien proponer un nuevo tema, o bien solicitar autorización para participar en el desarrollo de alguno ya propuesto.

Independientemente del tema elegido, cada usuario debe desarrollar sus páginas utilizando cualquier editor HTML. No creímos conveniente implementar uno, pues por un lado, no hace a la finalidad de la tesis, y por otro, existen muchos editores de muy buena calidad que son de fácil acceso para los usuarios. Las páginas logradas deben ser trasladadas a un directorio público de la máquina host de AMCO para poder ser compartidas

Cada una de las páginas que el usuario quiera proponer como página final de un tema, debe ser dada de alta en el sistema, para así exponerla a votación del resto de los integrantes del grupo.

Cuando se da de alta una página, el sistema internamente genera una copia de ella, a fin de preservarla de futuros cambios que puedan ser realizados ya sea por el propio autor o por otros.

En cuanto a la votación mencionada, se utiliza un mecanismo de decisión basado en un sistema al que podemos llamar "democrático", ya que si bien se apoya en la estrategia de decisión por mayoría (sección 3.2.3), no es tan rígida, debido al uso de porcentajes determinados lógicamente. Además, no se descarta la posibilidad de lograr una decisión por concenso (sección 3.2.3), pues los integrantes del grupo pueden discutir el tema en cuestión mediante el uso del chat, hasta llegar a una desición que satisfaga a todos. La toma de desiciones en AMCO se corresponde con el modelo racional (sección 3.2.1.1), ya que cada integrante de un grupo tiene en claro los objetivos del tema a realizar, y en base a ello, puede evaluar si la página que va a votar cumple con dichos objetivos. La teoría relacionada con la toma de decisiones está explicada en el capítulo 3.

Una página puede ser votada por todos aquellos usuarios que sean autores del tema al que pertenece la misma. Entonces, un usuario es autor de un tema cuando él lo propone o cuando, luego de haber pedido permiso para serlo, los demás autores lo aceptan como tal, mediante un sistema de votaciones equivalente al usado para determinar cuál de las páginas será la definitiva del tema al que pertenece.

AMCO, además, provee utilidades que complementan la funcionalidad del sistema, relacionadas con usuarios y páginas, entre ellas:

- Poder modificar los datos de un usuario, en caso de la existencia de errores, o de la necesidad de ampliación de la propia referencia.

- Conocer en qué situación se encuentra un usuario frente a un tema, es decir, saber si ya es o no autor del mismo.

7.3.2.- Chat

Como ya nombramos en una sección anterior, implementamos un Chat para el ambiente AMCO. La finalidad de esta implementación es la de posibilitar a los usuarios del ambiente poder discutir todo lo referente a la construcción y elección de páginas, como así también a la elección de los autores que las desarrollan. Esto no implica que el Chat no pueda utilizarse con su funcionalidad básica, que es la de permitir el intercambio de mensajes entre varias personas que se encuentren en lugares geográficos diferentes.

Cada usuario que se conecta al Chat tiene la posibilidad de escribir un mensaje, el que será enviado al resto de los usuarios conectados en ese momento. Cada uno de ellos podrá visualizar el mensaje en un espacio de texto, que muestra a dicho mensaje asociado al nombre del usuario que lo envió.

7.3.3.- Visor

El Visor tiene como finalidad permitir a cada uno de los usuarios de AMCO, conocer el estado en que se encuentra el resto de ellos. Esto se logra al visualizar la lista de los usuarios que están chateando, y la lista de los que están utilizando el Manejador de Páginas. Dichas listas se actualizan en tiempo real con cada entrada o salida de una nueva persona, a cada una de estas herramientas.

7.4.- Colaboración

Podemos afirmar que AMCO cumple con los principios de la colaboración, ya que por un lado corre sobre una plataforma Internet, la cual de por sí puede verse como una herramienta para colaborar, según lo expuesto en la sección 5.8.2.7; y por otro lado satisface los requerimientos que permiten configurar aplicaciones colaborativas / cooperativas (sección 1.6), y construír sistemas distribuidos eficientes, seguros y amigables, a saber:

- *Comunicación:* se provee una herramienta de comunicación textual, en tiempo real, que es el chat, y un mecanismo de votaciones, el cual puede verse como una forma de comunicación asincrónica, si consideramos que cada persona está emitiendo una opinión de algo que hizo otra cada vez que vota a favor o en contra de una página o de un aspirante a ser autor de un tema determinado.

- *Mecanismo de resolución de conflicto:* el mecanismo utilizado es un sistema de votaciones al que llamamos "democrático", que como lo explicamos en la sección 7.3.1, utiliza una estrategia de decisión por mayoría (sección 3.2.3), atenuada por el uso de porcentajes determinados lógicamente. Además, no se descarta la posibilidad de lograr una decisión por concenso (sección 3.2.3), pues los integrantes del grupo pueden discutir el tema en cuestión mediante el uso del chat, hasta llegar a una desición que satisfaga a todos.

- *Conocimiento del grupo:* Se permite conocer los datos referentes a los usuarios que quieren ser autores de un tema, en el momento en que se está frente a la pantalla de votación de aspirantes. Además, se permite conocer el estado en que se encuentran los usuarios del sistema mediante la herramienta visor, desarrollada para tal fin.

- *Proveer acceso, uso y compartimiento de información:* La forma de compartir información (páginas) para AMCO es externa al ambiente propiamente dicho, ya que tiene lugar gracias a que todos los usuarios pueden acceder a un directorio público de la máquina host, donde se encuentran todas las páginas desarrolladas. De esta forma cualquier usuario puede obtener una copia, modificar, eliminar, etc. páginas. Sin embargo, estas modificaciones no se reflejan en AMCO automáticamente, sino a través de una nueva alta, con un nuevo nombre, en el sistema. Esto no significa que se reemplacen las páginas, sino que las versiones anteriores se mantienen para el proceso de selección de la página definitiva.

- *Mecanismo para búsqueda y acceso de la información:* AMCO tiene un único mecanismo de búsqueda y acceso de la información: estructurada por temas.

- *Interfaces amigables:* El sistema está construído en base a una serie de ventanas independientes unas de otras, que el usuario puede acomodar en la pantalla de la manera más conveniente para su mejor uso. La cantidad de ventanas visibles en un momento determinado de la utilización del sistema queda a criterio del usuario.

- *Control de acceso:* La única herramienta que provee control de acceso dentro de AMCO es el Manejador de Páginas, el que requiere del ingreso de una password a fin de identificar unívocamente al usuario, y así evitar confundirlo con otros integrantes que tengan el mismo nombre.

Los usuarios del ambiente tienen la posibilidad de colaborar sincrónica o asincrónicamente, dependiendo de la parte de la aplicación que se esté utilizando. La colaboración sincrónica se refleja mediante el uso del chat, en el que los participantes interactúan simultáneamente, a pesar de estar geográficamente dispersos. El resto del ambiente implica colaboración asincrónica, pues los usuarios tienen la libertad de ejecutar cualquier tarea, en cualquier momento. Esto significa que dos usuarios pueden trabajar en diferentes momentos, o, pueden trabajar en el mismo momento, en la misma tarea o en tareas distintas. La teoría detallada de este tema se encuentra en la sección 1.5.

Por otra parte, podemos afirmar que AMCO es una aplicación CSCW (tema desarrollado en el capítulo 6), pues satisface sus características claves, entre ellas:

- *Trabajo en grupo:* Construcción colaborativa de páginas HTML.

- *Interfaces multiusuario:* Interfaces disponibles a todos los usuarios a través de Internet.

- *Trabajo en forma concurrente:* Presencia de comunicación sincrónica (Chat), y asincrónica (Manejador de Páginas).

- *Comunicación entre los integrantes del grupo:* Por medio del Chat y del sistema de votaciones.

- *Espacios de información compartida:* Espacio común de información a partir del cual se pueden guardar y recuperar las páginas.

AMCO intenta seguir los pasos de Groupware desde su perspectiva orgánica, ya que combina comunicación, colaboración y coordinación, aunque en forma muy elemental. Decimos que:

- hay comunicación (sección 5.8.1) porque los integrantes de los grupos pueden entablar diálogos a través del Chat;

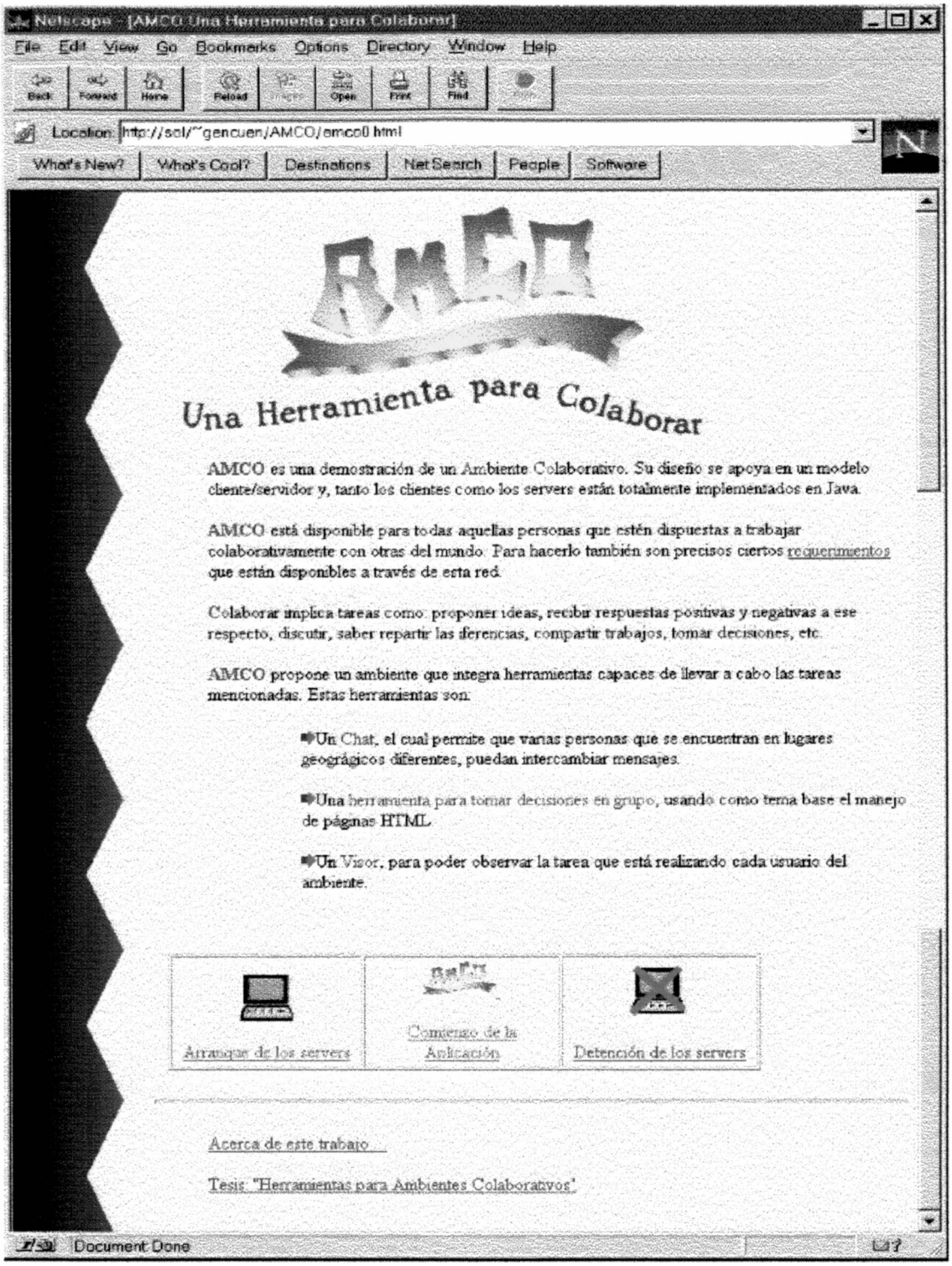
- hay colaboración (sección 5.8.2) porque desde el punto de vista general, la propia Web es una herramienta colaborativa, y desde el punto de vista de la aplicación en sí, todos los usuarios que son autores de un tema pueden compartir, discutir y votar páginas HTML realizadas por ellos; y

- hay coordinación (sección 5.8.3) en el sentido básico de llevar a cabo tareas en una secuencia determinada, por ejemplo: sólo se pueden votar páginas, si éstas previamente fueron creadas y siempre que quien vaya a votar sea autor del tema, lo que implica que antes el resto de los integrantes del grupo lo hayan aceptado como tal.

7.5.- Visualización y Explicación

7.5.1.- Pantalla Inicial

El ambiente AMCO está disponible para todos los usuarios de Netscape 3.0.1, bajo los requerimientos que se detallarán en el capítulo siguiente. A su pantalla de presentación se llega mediante la URL: http://www-lifia/AMCO/index.html, y su formato es:



Esta es la pantalla de presentación de AMCO. En ella se describen sus cterísticas generales, a fin de que los usuarios que la visiten logren obtener una idea al de las tareas que se pueden desarrollar en el ambiente. También contiene:

cono Habilitar Server: ejecuta un programa CGI, programado en C, que activa los ervers: registry de RMI, ServerInterf, ServerChat y ServerGenPag, cuya ıncionalidad es explicada en el próximo capítulo.

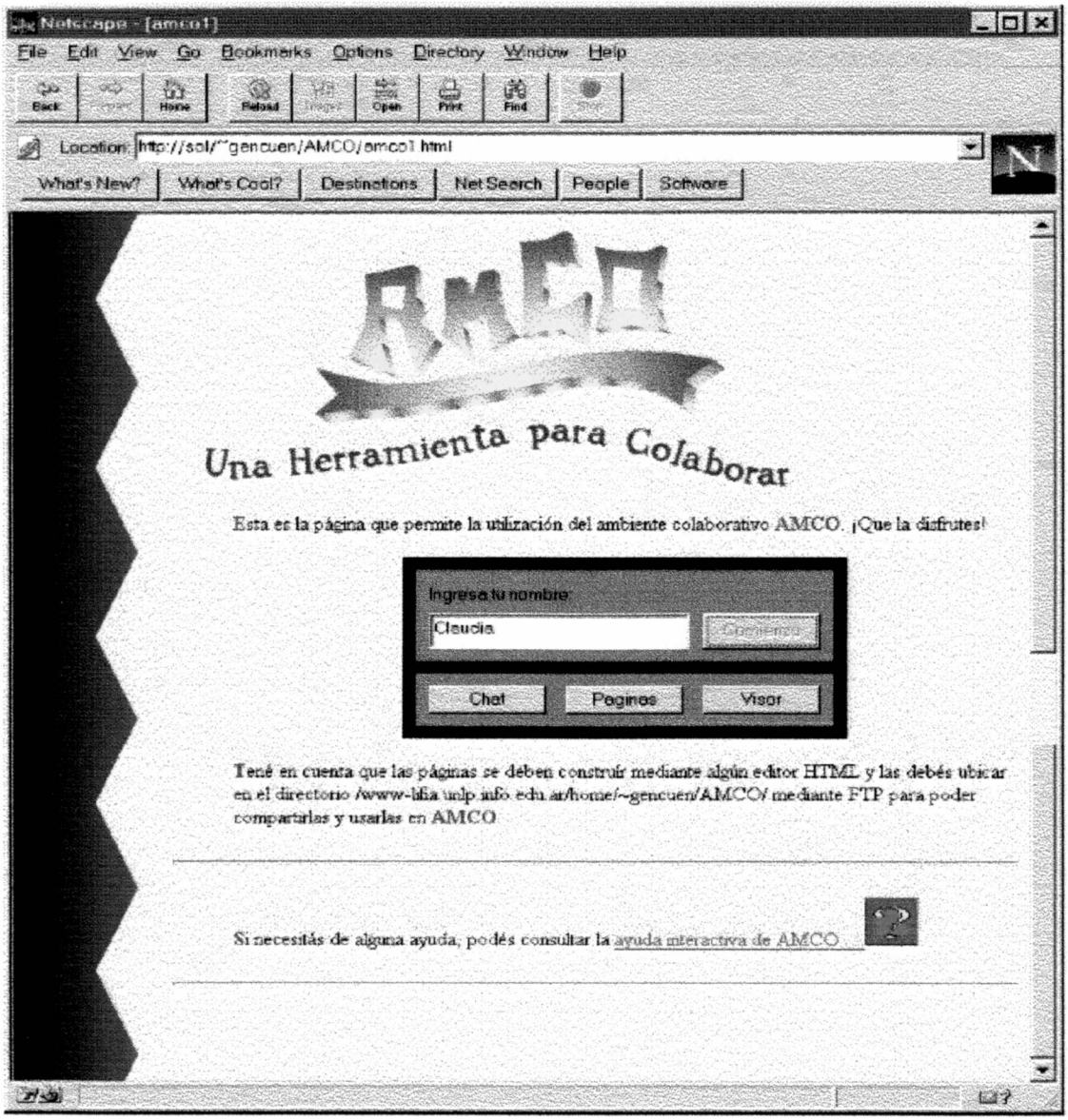
Icono deshabilitar server: ejecuta un programa CGI, programado en C, que desactiva los servers: registry de RMI, ServerInterf, ServerChat y ServerGenPag. El uso de este botón para la finalización de los servers tiene la misma finalidad que en el caso del inicio de ellos.

Link a requerimientos: Este link permite la navegación hacia otra página HTML donde se da a conocer el software necesario para usar AMCO.

Icono link: Este ícono permite la navegación hacia otra página HTML, la cual contiene la pantalla principal del ambiente AMCO.

5.2.- Pantalla Principal

Esta pantalla surge a partir del *Icono Link*. Su formato es:



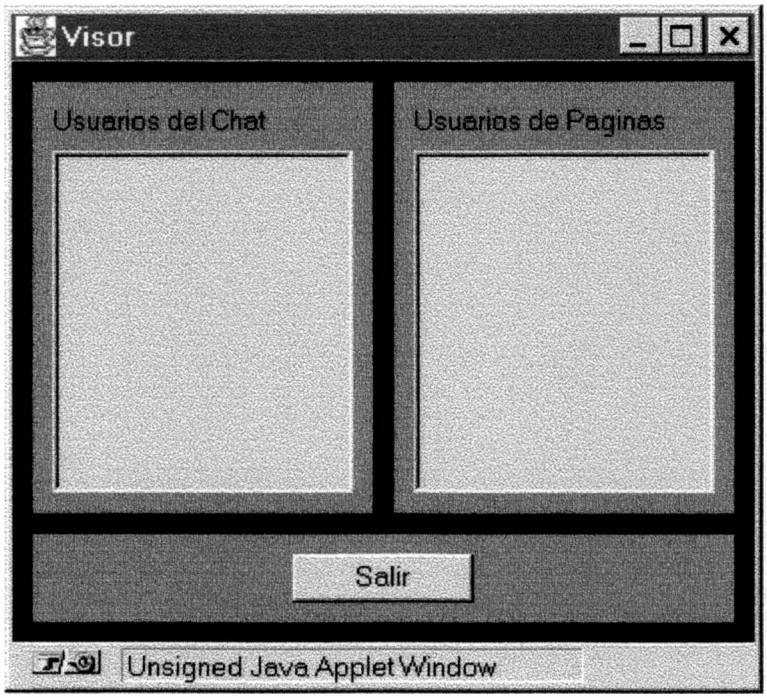
Esta pantalla contiene:

- *Una ayuda del ambiente:* Esta ayuda permitirá a los usuarios conocer la funcionalidad de cada una de las ventanas de AMCO. Para ello, a partir de la ventana principal se podrá navegar a cada una de sus opciones, obteniéndose así información visual y textual de ellas, a fin de que el usuario pueda entender mejor el funcionamiento del sistema colaborativo.

- *El ambiente AMCO en sí mismo*: Permite a los usuarios el inicio de la sesión colaborativa. Dicha sesión requiere, como primera medida, el ingreso del nombre del usuario, el que quedará registrado en el sistema luego de presionar el botón "COMIENZO", que es el único activo hasta el momento. Acto seguido, se deshabilita dicho botón y quedan disponibles al usuario las tres herramientas que provee este ambiente, a las cuales se puede acceder presionando los botones: Visor, Chat y Páginas, que despliegan las ventanas Visor, Chat y Manejador de Páginas, respectivamente.

7.5.2.1.- Ventana Visor

Esta ventana surge a partir del *Botón Visor*. Su formato es:



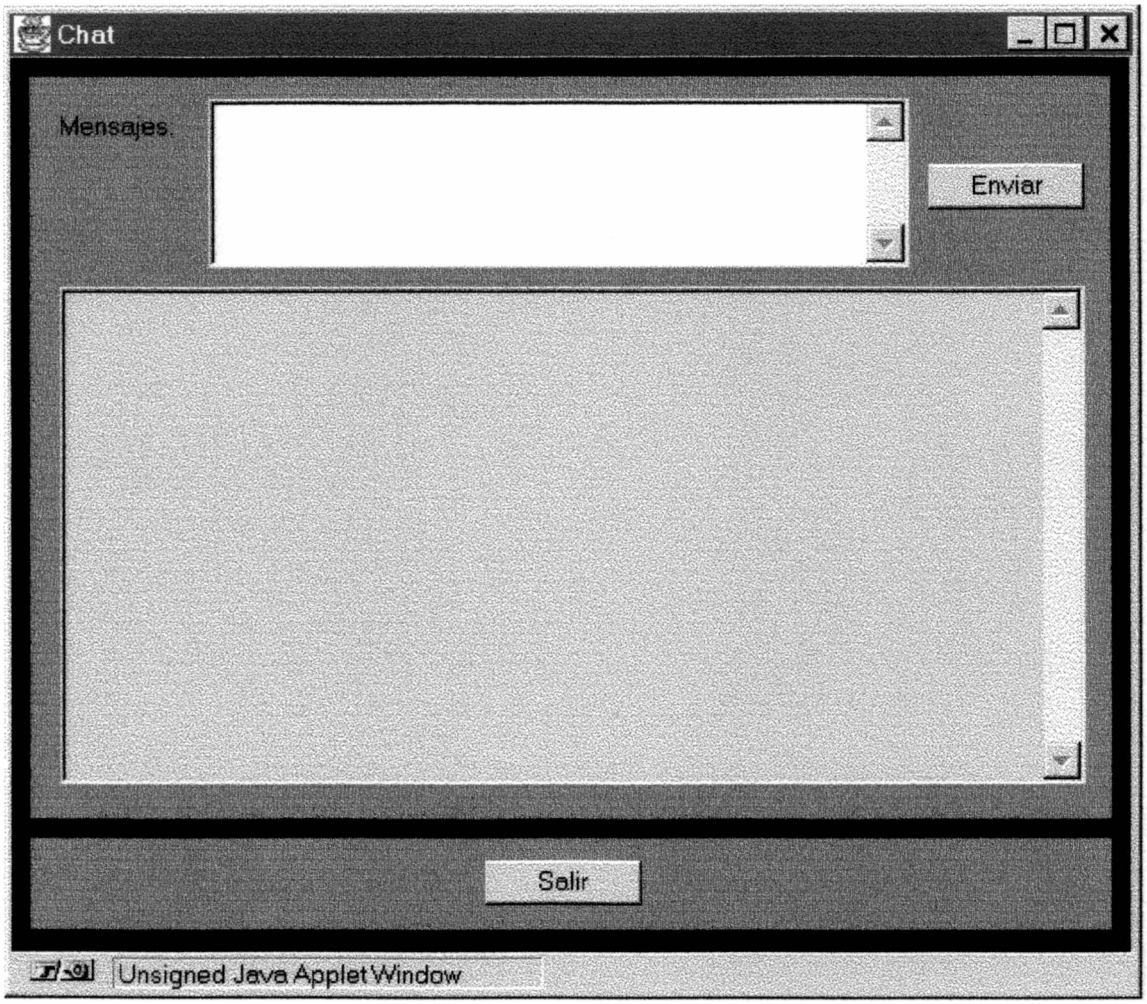
Esta ventana consta de dos sectores: uno con la lista de los nombres de los usuarios que en ese momento están utilizando el Chat, y otro, con la lista de los nombres de los usuarios que en ese momento están utilizando el Manejador de Páginas.

Si un usuario comienza o abandona una sesión del Chat y/o del Manejador de Páginas mientras la ventana del Visor está visible, ésta se actualiza inmediatamente.

El Visor también cuenta con un botón Salir, cuya única finalidad es ocultar la ventana.

.5.2.2.- Ventana Chat

Esta ventana surge a partir del *Botón Chat*, cuya funcionalidad es incorporar al suario a la lista de personas que están chateando en el sistema y desplegar la siguiente entana:

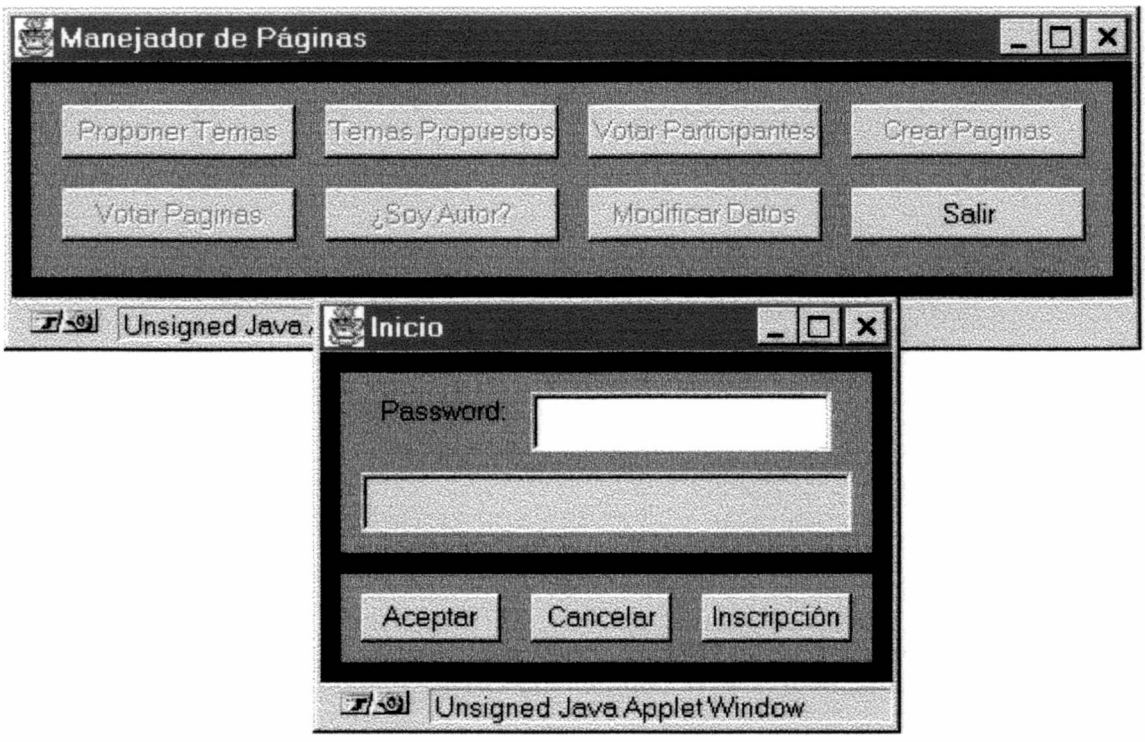


Esta ventana brinda la interface apropiada para que los usuarios conectados al istema puedan intercambiar sus mensajes. Para ello se cuenta con:

- *Un sector ("Mensajes")* para el ingreso del texto del mensaje a enviar al resto de los usuarios conectados en ese momento.
- *Un botón Enviar*: Este botón es el que efectiviza el envío del mensaje.
- *Un área de texto* donde se mostrarán los mensajes recibidos por los usuarios conectados al sistema. Cada mensaje tendrá el siguiente formato:
  NOMBRE_DEL_USUARIO : TEXTO_DEL_MENSAJE.
- *Un botón Salir*: Este botón ocultará la ventana y eliminará al usuario de la lista de personas que están chateando en el sistema.

7.5.2.3.- Ventana Manejador de Páginas

Esta ventana surge a partir del *Botón Páginas*, cuya funcionalidad es dar comienzo a la herramienta colaborativa de manipulación de páginas y desplegar la ventana principal y la ventana de inicio de sesión.

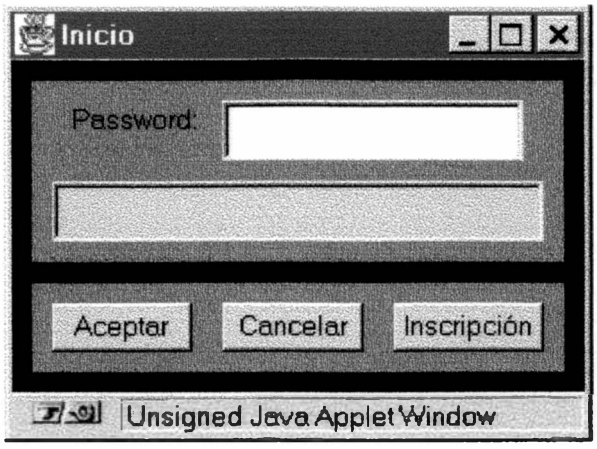


La ventana Principal tendrá todos los botones desactivados hasta que el usuario indique su presencia en la herramienta mediante la ventana de Inicio. Esta registración es necesaria, independientemente de la efectuada en la pantalla principal de AMCO, debido a que únicamente en esta herramienta se precisa de una password. El objetivo de disponer de dicha password es identificar unívocamente a los usuarios, de manera de poder asociar y/o recuperar las páginas, temas, etc. que le corresponden a cada uno.

Dicha asociación y/o recuperación no podría realizarse de otra forma, por ejemplo: por medio del nombre del usuario, ya que no se sabría cuándo se trata del mismo usuario, y cuándo de un usuario distinto con el mismo nombre.

7.5.2.3.1.- Ventana de Inicio

Esta ventana es desplegada a partir del *Botón Páginas*, junto con la ventana Principal del Manejador de Páginas. Su formato es el siguiente:

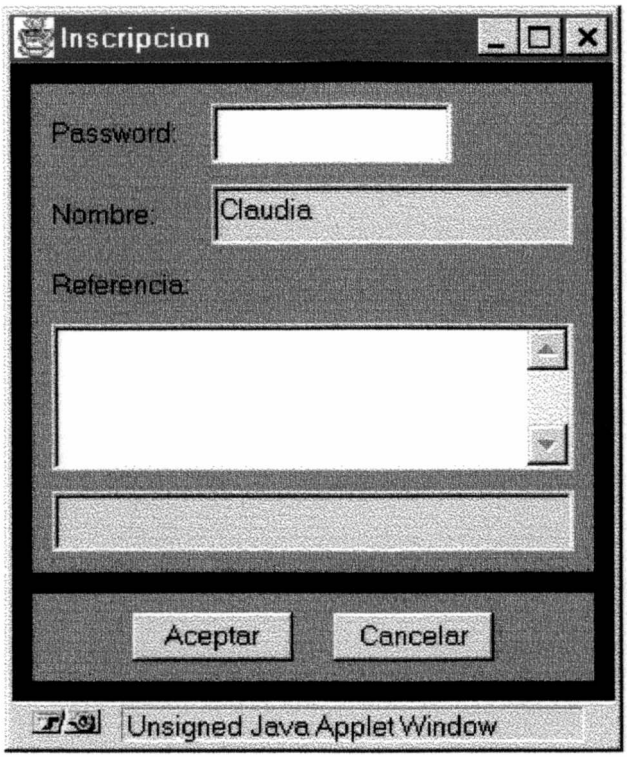


Esta ventana permite registrar la presencia del usuario en el Manejador de Páginas, paso necesario para el ingreso de la password. Consta de:

- *Un sector ("Password")* para ingresar la password del usuario.
- *Un sector de texto* reservado para comunicar al usuario los mensajes necesarios, por ejemplo: si la password es o no válida.
- *Un botón Aceptar:* Una vez que la password haya sido ingresada, el programa inmediatamente determina si es válida, es decir, si dicha password pertenece a algún usuario ya inscripto en el sistema. En este caso, la ventana se oculta y se habilitan los botones de la ventana Principal. Si la password no es válida, significa que el usuario aún no ha trabajado en el sistema y debe inscribirse, ésto le será informado en el sector de texto reservado para tal fin.
- *Un botón Cancelar:* Este botón implica la salida de la herramienta Manejador de Páginas, ocultando tanto la ventana de Inicio como la ventana Principal.
- *Un botón Inscripción:* Este botón muestra una ventana cuya funcionalidad es incorporar a los usuarios al sistema. Esto significa que la herramienta guarda la información del usuario de manera tal que, en próximas sesiones, este usuario sólo necesite ingresar la password para tener acceso al sistema.

7.5.2.3.1.1.- Ventana de Inscripción

Esta ventana es desplegada a partir del *Botón Inscripción,* de la ventana Inicio. Su formato es:

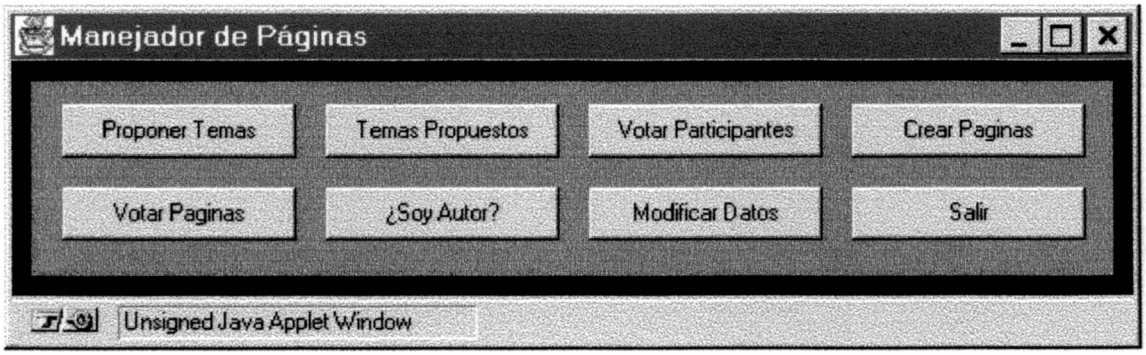


La funcionalidad de esta ventana es la de registrar a un usuario como integrante del Manejador de Páginas. Este paso debe ser cumplido por cada usuario, únicamente la primera vez que ingresa a la herramienta. En posteriores ocasiones, alcanza con informar su password. La ventana consta de los siguientes sectores:

- *Un sector ("Password")* para el ingreso de la password del usuario.
- *Un sector ("Nombre")* que muestra el nombre del usuario, ingresado en la pantalla principal de AMCO.
- *Un sector ("Referencia")* para el ingreso de las referencias y demás datos que el usuario quiera dar a conocer al resto de los usuarios.
- *Un sector de texto* reservado para comunicar mensajes al usuario.
- *Un botón Aceptar*: Este botón verifica que todos los datos hayan sido ingresados y si es así, se asegura que la password no pertenezca a ningún usuario ya inscripto en el sistema. En el caso en que todo esté correcto, el usuario se incorpora al sistema y se oculta la ventana, devolviéndose el control a la ventana de Inicio para, ahora sí, ingresar la password y dar comienzo a la sesión del Manejador de Páginas.
- *Un botón Cancelar:* Su única finalidad es ocultar la ventana y retornar el control a la ventana de Inicio.

7.5.2.3.2.- Ventana Principal

Esta ventana es desplegada a partir del *Botón Páginas*, de la ventana Inicial de AMCO. Su funcionalidad sólo estará disponible una vez que se haya dado comienzo a la sesión, por medio de la ventana de Inicio. Su formato es:

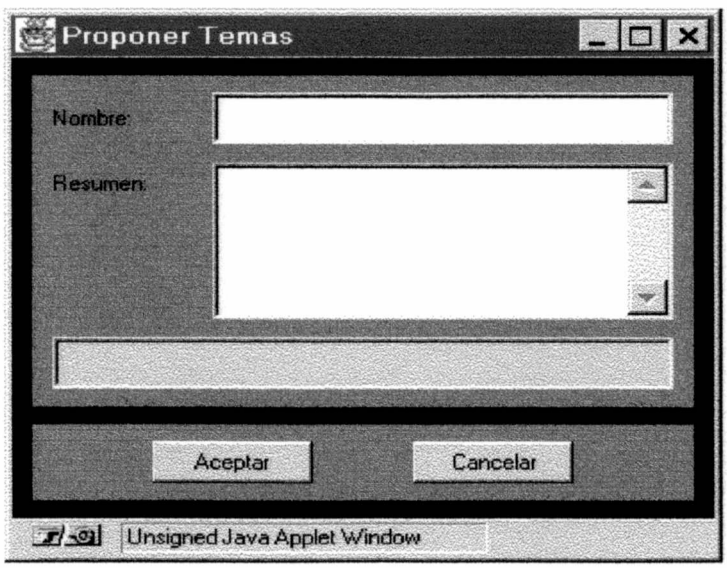


Esta ventana consta de ocho botones. La funcionalidad del conjunto de ellos permite el manejo colaborativo de páginas HTML:

- *Botón Proponer Tema:* Despliega la ventana Proponer Temas.
- *Botón Temas Propuestos*: Despliega la ventana Temas Propuestos.
- *Botón Votar Aspirante:* Despliega la ventana Votar Aspirante.
- *Botón Crear Páginas:* Despliega la ventana Crear Páginas.
- *Botón Votar Páginas:* Despliega la ventana Votar Páginas.
- *Botón ¿Soy Autor?:* Despliega la ventana ¿Soy Autor?.
- *Botón Modificar Datos:* Despliega la ventana Modificar Datos.
- *Botón Salir:* Oculta la ventana.

7.5.2.3.2.1.- Ventana Proponer Temas

Esta ventana es desplegada a partir del *Botón Proponer Temas*, de la ventana Principal del Manejador de Páginas. Su formato es:



Esta ventana permite la especificación de un tema que quedará registrado en la herramienta para que otros usuarios puedan asociarse a él y así, posibilitar la creación conjunta de páginas alrededor de ese tema. Consta de:

- *Un sector ("Nombre")* para el ingreso del nombre del tema.
- *Un sector ("Resumen")* para el ingreso de la información asociada a ese tema.
- *Un sector de texto* reservado para comunicar mensajes al usuario.
- *Un botón Aceptar:* Este botón verifica que se hayan ingresado tanto el nombre del tema como su descripción, para así poder efectivizar el alta de dicho tema. Con ésto se logra su incorporación a la lista de temas propuestos, que queda disponible al resto de los usuarios. También registra a este usuario como autor del tema.
- *Un botón Cancelar:* Su única funcionalidad es ocultar la ventana.

7.5.2.3.2.2.- Ventana Temas Propuestos

Esta ventana es desplegada a partir del *Botón Temas Propuestos,* de la ventana Principal del Manejador de Páginas. Su formato es:

Esta ventana permite a todos los usuarios del sistema, recorrer los temas propuestos. El primero de ellos se visualiza inmediatamente a que la ventana sea mostrada. Consta de:

- *Un sector ("Nombre")* donde se muestra el nombre del tema propuesto.
- *Un sector ("Resumen")* donde se muestra la descripción del tema propuesto.

- *Un sector de texto* reservado para comunicar mensajes al usuario.

- *Un botón Anterior*: Este botón muestra los datos del tema anterior al visualizado en ese momento, de la lista de temas.

- *Un botón Siguiente:* Este botón muestra los datos del tema siguiente al visualizado en ese momento, de la lista de temas.

- *Un botón Permiso:* Este botón agrega al usuario a la lista de aspirantes a ser autores del tema seleccionado, siempre que aún no lo sea.

- *Un botón Salir:* Su única finalidad es ocultar la ventana.

7.5.2.3.2.3.- Ventana Votar Aspirante

Esta ventana es desplegada a partir del *Botón Votar Aspirantes*, de la ventana Principal del Manejador de Páginas. Su formato es:

Esta ventana permite votar a favor o en contra de un aspirante para que éste pueda transformarse o no en autor de un tema determinado. Consta de:

- *Un sector ("Lista de Temas")* donde se muestra la lista de todos los temas para los cuales el usuario es autor.

- *Un sector ("Lista de Participantes")* donde se muestra la lista de todos los aspirantes de un tema determinado.

- *Un sector ("Referencias")* donde se muestran los datos de un aspirante determinado.

En esta ventana, inicialmente, sólo se muestra la lista de temas del autor. Una vez seleccionado uno de ellos, se muestra automáticamente la lista de aspirantes para ese tema. Y, una vez seleccionado un aspirante, se muestran sus datos.

- *Un sector de texto* reservado para comunicar mensajes al usuario.

- *Un botón Votar a Favor*: Presionar este botón implica que el usuario acepta al aspirante seleccionado como autor del tema elegido.

- *Un botón Votar en Contra*: Presionar este botón implica que el usuario no acepta al aspirante seleccionado como autor del tema elegido.

- *Un botón Salir:* Su única funcionalidad es ocultar la ventana.

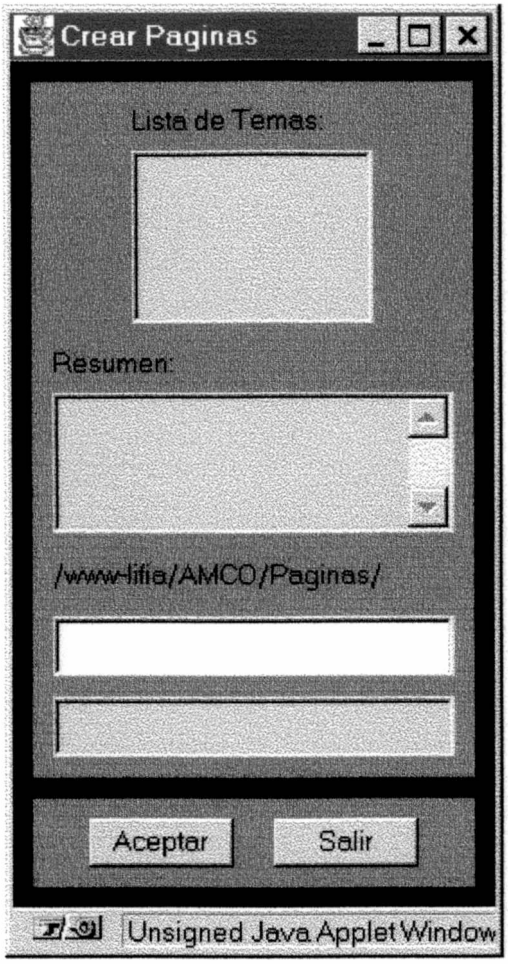
Cada vez que un usuario vota a favor de un aspirante de un tema determinado, se incrementa en uno la cantidad de votos a favor de dicho aspirante y la cantidad de autores que lo votaron. En el caso del voto en contra, sólo se incrementa en uno la cantidad de autores que lo votaron.

Ya sea cuando se realiza un voto a favor, como cuando se realiza uno en contra, se verifica qué porcentaje representa la cantidad de votos a favor con respecto a la cantidad de autores que votaron. Si este porcentaje es mayor al 70 %, el aspirante pasa a ser autor del tema. En el caso en que hayan votado el 85 % de los autores y la cantidad de votos a favor del aspirante en cuestión no supere al 70 %, entonces este aspirante ya no tiene posibilidades de ser autor del tema, por lo que automáticamente es eliminado de la lista de aspirantes a ese tema. Esta eliminación tiene como objetivo la eficiencia del sistema de votaciones, evitando así el testeo de aspirantes que ya no tienen posibilidades de llegar a ser autores de un tema.

La determinación de los porcentajes explicados más arriba, fue una decisión que tomamos basada en el sentido común, para atenuar la rigidez de la estrategia de votos por mayoría. La justificación de esta determinación fue expresada en la sección 7.3.1.

7.5.2.3.2.4.- Ventana Crear Páginas

Esta ventana es desplegada a partir del *Botón Crear Páginas*, de la ventana Principal del Manejador de Páginas. Su formato es:



Esta ventana permite dar de alta a una nueva página, asociada a un tema determinado. Esto significa que dicha página quedará a disposición de los demás autores del tema al que pertenence a fin de que puedan emitir su voto. Consta de:

- *Un sector ("Lista de Temas")* donde se muestra la lista de todos los temas para los cuales ese usuario es autor.

- *Un sector ("Resumen")* donde, una vez seleccionado un tema, automáticamente se muestran los datos asociados a él.

- *Un sector* donde se debe ingresar el nombre de la página que se quiere dar de alta al sistema. Este nombre debe representar a un archivo .html que se encuentre ubicado a partir del directorio "/www-lifia/AMCO/Paginas/". La explicación detallada de este requerimiento se encuentra en la sección 7.3.1.

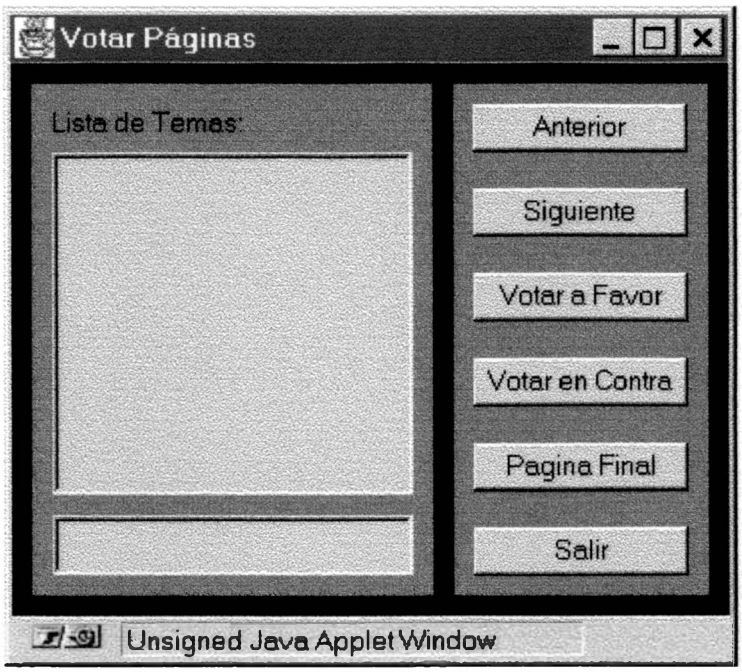
- *Un sector de texto* reservado para comunicar mensajes al usuario.

- *Un botón Aceptar*: Este botón, como primera medida, verifica que el nombre de la página ingresada exista, si es así, la da de alta. En el caso en que dicha página ya pertenezca a la lista de páginas asociadas al tema seleccionado, o que el archivo correspondiente al nombre ingresado no sea encontrado, se le informará al usuario en el espacio de texto reservado para tal fin.

- *Un botón Salir:* Su única finalidad es ocultar la ventana.

7.5.2.3.2.5.- Ventana Votar Páginas

Esta ventana es desplegada a partir del *Botón Votar Páginas,* de la ventana Principal del Manejador de Páginas. Su formato es:



Esta ventana permite visualizar todas las páginas asociadas a un tema determinado y votar a favor o en contra de la página visible en ese momento. Consta de:

- *Un sector ("Lista de Temas")* donde se muestra la lista de todos los temas para los cuales el usuario es autor. Una vez seleccionado un tema, se abre automáticamente una ventana Netscape mostrando la primer página asociada al tema seleccionado. Esto permite que el usuario pueda visualizar las páginas en el momento de efectuar la correspondiente votación.

- *Un botón Anterior*: Este botón muestra la página anterior de la lista de páginas asociada a un tema seleccionado.

- *Un botón Siguiente*: Este botón muestra la página siguiente de la lista de páginas asociada a un tema seleccionado.

- *Un botón Votar a Favor:* Presionar este botón implica que el usuario está de acuerdo en incluír a la página visible en ese momento, en la lista de candidatas a ser la definitiva del tema seleccionado.

- *Un botón Votar en Contra:* Presionar este botón implica que el usuario no está de acuerdo en incluír a la página visible en ese momento, en la lista de candidatas a ser la definitiva del tema seleccionado.

- *Un botón Pagina Final:* Este botón abre una ventana Netscape permitiendo la visualización de la página que tiene la mayor cantidad de votos a favor del tema seleccionado, si es que ya se han efectuado votaciones para ella. En caso contrario, se informa al usuario en el sector de texto reservado para tal fin.

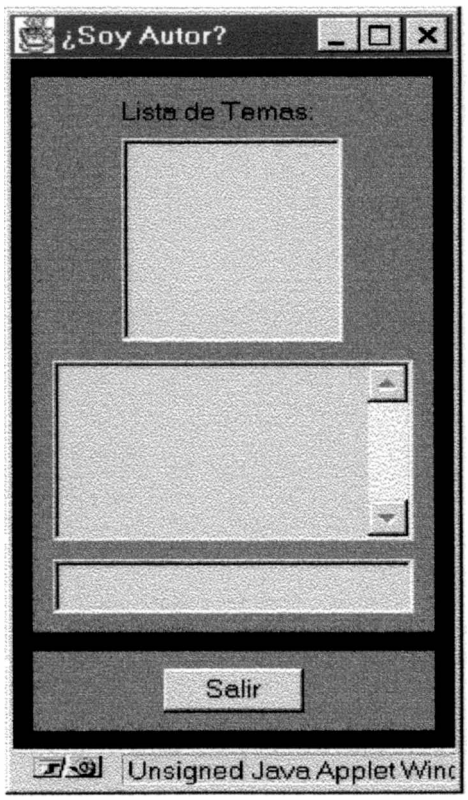
- *Un botón Salir:* Su única funcionalidad es ocultar la ventana.

Cada vez que un usuario vota a favor de una página de un tema determinado, se incrementa en uno la cantidad de votos a favor de esa página y la cantidad de autores que la votaron. En el caso del voto en contra, sólo se incrementa en uno la cantidad de autores que la votaron.

Ya sea cuando se realiza un voto a favor, como cuando se realiza uno en contra, se verifica qué porcentaje representa la cantidad de votos a favor con respecto a la cantidad de autores que votaron. Si este porcentaje es mayor al 70 %, y a su vez es el mayor entre las páginas asociadas al tema seleccionado, dicha página pasa a ser la Final del mismo. Si hubiera un empate, se debe verificar cuál de todas las páginas que empataron tiene mayor porcentaje de votación por parte de los autores, y esa pasa a ser la página Final. Si también hubiera empate, no se efectúan modificaciones. Si una página tiene el 85 % de los votos de los autores y su porcentaje de aceptación es menor al de la página Final, será eliminada automáticamente. Esta eliminación tiene como objetivo la eficiencia del sistema de votaciones, evitando así el testeo de páginas que ya no tienen posibilidades de llegar a ser la definitiva.

7.5.2.3.2.6.- Ventana ¿Soy Autor?

Esta ventana es desplegada a partir del *Botón ¿Soy Autor?*, de la ventana Principal del Manejador de Páginas. Su formato es:



Esta ventana permite a un autor obtener información acerca de su estado con respecto a un tema determinado, es decir saber si ya es autor o no del mismo. Consta de:

- *Un sector ("Lista de Temas")* donde se muestra la lista de todos los temas que tienen relación con el usuario, es decir, la suma de los temas para los cuales es autor y para los que aún es aspirante.

- *Un sector de texto* reservado para comunicar mensajes al usuario.

Una vez seleccionado un tema de las lista de temas, el programa internamente verifica si el usuario es autor o no de ese tema. Esta información se le da a conocer mediante el sector de texto reservado para tal fin, informándose: "Autor", en el caso de ser autor del tema seleccionado, e "Indefinido" en caso contrario.

- *Un botón Salir:* Su única funcionalidad es ocultar la ventana.

7.5.2.3.2.7.- Ventana Modificar Datos

Esta ventana es desplegada a partir del *Botón Modificar Datos*, de la ventana Principal del Manejador de Páginas. Su formato es:

Esta ventana permite al usuario realizar modificaciones en su password, nombre y/o datos personales. Consta de:

- *Un sector ("Password")* donde se muestra la password actual del usuario y se permite su modificación.

- *Un sector ("Nombre")* donde se muestra el nombre actual del usuario y se permite su modificación.

- *Un sector ("Referencias")* donde se muestran los datos actuales del usuario y se permite su modificación.

- *Un sector de texto* reservado para comunicar mensajes al usuario.

- *Un sector de elección* del dato o datos que se quieran modificar, es decir se puede cambiar uno sólo o cualquier combinación de ellos.

- *Un botón Aceptar:* Este botón realiza las modificaciones de acuerdo a los datos seleccionados en el sector de elección. Si ninguna de las opciones fue marcada, la modificación queda sin efecto. Luego, se oculta la ventana.

- *Un botón Cancelar:* Su única finalidad es ocultar la ventana.

Capítulo 8

AMCO: Una Herramienta para Colaborar - Visión Interna

Este capítulo estará dividido en dos secciones:

- *Sección I*: Contendrá los temas: Objetos Distribuidos, Java y el paquete RMI (Remote Method Invocation), que es el modelo de objetos distribuido de Java;

- *Sección II*: Contendrá la especificación de estos temas aplicados a AMCO.

Sección I

8.1.- Objetos Distribuidos [29]

8.1.1.- Ideas Generales

Los objetos distribuidos son un ambiente de computación que resultó de la convergencia de las tecnologías orientada a objetos y cliente/servidor. Combinan las ventajas de la distribución de la tecnología cliente/servidor, con la riqueza de información del mundo real, contenida en los modelos orientados a objetos.

La distribución de objetos es un paradigma de computación que permite a los objetos estar distribuidos a lo largo de una red heterogénea, y permite a cada uno de los componentes interoperar como un todo unificado. Para una aplicación construída en un ambiente orientado a objetos, la red es la computadora.

Los objetos interactúan pasándose mensajes unos a otros. Estos mensajes representan pedidos de información o servicio. Durante una determinada interacción, los objetos dinámicamente asumirán los roles de clientes y servers. Lo que físicamente une a los objetos distribuidos es un intermediario de pedido de objetos (Object Request Broker = ORB). El ORB provee los medios para que el objeto localice y active a otros objetos en la red, sin tener en cuenta el procesador o lenguaje de programación usado para desarrollar los objetos cliente o server. El ORB hace que estas cosas ocurran en forma transparente para el desarrollador. Además, es quien permite la interoperabilidad en redes de objetos heterogéneas. Permite también la localización, activación y comunicación con objetos, al mismo tiempo que oculta al desarrollador sus detalles de implementación. Su gran ventaja es que elimina la complejidad del pasaje de mensajes a través de la red.

Cuando los objetos están distribuidos a lo largo de una red, los clientes pueden ser servers y recíprocamente, los servers pueden ser clientes. Esto realmente no importa ya que estamos tratando con objetos que cooperan: el cliente pide servicios de otro objeto y el objeto server satisface el pedido. Clientes y servers pueden estar físicamente

en distintos lugares de la red y estar escritos en diferentes lenguajes de programación orientados a objetos. Aunque los clientes y servers universales viven en mundos dinámicos propios fuera de una aplicación, los objetos parecen ser locales a la aplicación debido a que la red es la computadora. En esencia, el concepto completo de objetos distribuidos puede verse simplemente como una red global de clientes y servers heterogéneos o, más precisamente, objetos cooperativos.

Los objetos sólo conocen qué servicios les proveen los otros objetos (sus interfaces), pero no cómo son provistos (su implementación). El ocultamiento de los detalles de implementación dentro de los objetos es una de las contribuciones claves de la tecnología orientada a objetos para manejar la complejidad de la computación distribuida. En un ambiente de objetos distribuidos, el desarrollador de la aplicación no tiene que considerar qué máquina o lenguaje de programación se usa para implementar los objetos servers. La visión del usuario de una aplicación consiste en objetos que, pueden estar escritos en C++ y correr en una máquina, o ser un programa COBOL que corra en un mainframe, o un objeto Smalltalk que corra dentro de la estación de trabajo de un usuario, o una planilla de cálculo Excel que corra en una microcomputadora. El usuario no tiene nada que ver con estas plataformas y ambientes de programación, simplemente ve objetos que interactúan unos con otros como un todo unificado.

En resumen, los objetos distribuidos son una extensión de la tecnología cliente/servidor. Sin embargo, hay una diferencia en su proceso de trabajo e implementación. Con el modelo cliente/servidor, hay generalmente una aplicación que corre en una computadora cliente, mientras otra corre en una computadora server. Estas dos aplicaciones se comunican a través de una red y relacionan sus datos, generalmente vía algún intermediario provisto en la forma de una interface de programa de aplicación (API) o librería de llamadas a funciones.

En un sentido, el modelo cliente/servidor es una versión restringida del modelo de objetos distribuidos. Una aplicación distribuida está hecha de objetos, de la misma manera que cualquier otra aplicación orientada a objetos. Sin embargo, los objetos de una aplicación de objetos distribuidos pueden dividirse y correrse sobre diversas computadoras a lo largo de una red.

8.1.2.- Ventajas técnicas de los objetos distribuidos

Los objetos distribuidos pueden simplificar radicalmente el desarrollo de aplicaciones. Los modelos y herramientas de objetos distribuidos extienden un sistema de programación orientada a objetos. Los objetos pueden estar distribuidos en diferentes computadoras a través de una red, viviendo en sus propias librerías dinámicas fuera de una aplicación y aún parecer estar locales a una aplicación. Existen varias ventajas técnicas del ambiente de objetos distribuidos:

- Las ventajas de la herencia pueden ser la base. Los objetos ocultados (interfaces orientadas a objetos que acceden a códigos heredados) pueden aplicarse a diferentes recursos de computación a través de una red, a fin de simplificar los medios de comunicación con esos recursos. Toda la comunicación entre objetos distribuidos ocurre en la forma de mensajes, de la misma manera en que se comunican los objetos locales dentro de una aplicación, más que aplicando diferentes intermediarios e interfaces de red a

cada sistema heredado. Además, un solo objeto puede representar información derivada de muchos sistema heredados.

- Debido a que todos los objetos (locales y remotos) se comunican de la misma forma (vía mensajes), los programadores tiene la habilidad para distribuir los componentes de una aplicación en las computadoras que mejor se adaptan a las tareas de cada objeto, sin tener que cambiar el resto de la aplicación que usa esos objetos. Por ejemplo, un objeto que realiza computaciones intensivas podría colocarse en una computadora más poderosa que una de escritorio promedio, donde el usuario interactúa con la presentación de los objetos.

- Debido a que los objetos aparentan ser locales a sus clientes, un cliente no conoce en qué máquina o tipo de máquina residen. Como resultado, la migración de objetos de implementación de una plataforma a otra es mucho más fácil, y puede realizarse por pasos sin afectar a los clientes.

- La integración de sistemas puede realizarse a un grado mayor. Los recursos de software y hardware disponibles en plataformas dispares pueden ser unidos en una sola aplicación. El objetivo de crear una sola imágen del sistema se logra cuando las aplicaciones pueden ser unidas a partir de objetos distribuidos.

El objetivo técnico completo de los objetos distribuidos es claro: mejorar la tecnología cliente/servidor de manera de ser más eficiente y flexible, y menos compleja. Los beneficios de la distribución de objetos son soluciones reales a los problemas con los paradigmas cliente/servidor existentes.

8.2.- Java [25]

Java es un lenguaje de programación simple, familiar, orientado a objetos, de arquitectura neutral, portable, robusto, interpretado, dinámico, seguro, multithreaded y de alta performance.

8.2.1 .- Simple y Familiar

Java muestra un nuevo punto de vista en la evolución de los lenguajes de programación: la creación de un lenguaje simple y pequeño que es lo suficientemente inteligente como para permitir el desarrollo de una gran variedad de aplicaciones. Aunque Java es superficialmente parecido a C y C++, consiguió su simplicidad de la eliminación sistemática de características de sus predecesores. Esta eliminación es la que mantiene a Java pequeño y la que reduce el peso de los programadores en cuanto a la producción de código confiable.

Otro objetivo del diseño de Java era resultarle familiar a los programadores de PC y estaciones de trabajo, para lo cual se tuvo en cuenta que la mayoría de ellos están familiarizados con C, C++, Eiffel, Ada y otros lenguajes relacionados, por lo que la curva en el aprendizaje de Java sería pequeña.

8.2.2 .- Orientado a Objetos

Como Java es considerado verdaderamente orientado a objetos, deberá soportar por lo menos estas cuatro características:

- *Encapsulamiento*: Implementa comportamiento y modularidad de información

- *Polimorfismo:* El mismo mensaje enviado a distintos objetos se comporta según la naturaleza del objeto que lo recibe.

- *Herencia:* Se definen nuevas clases y comportamiento en base a clases existentes para obtener código organizado y reusable.

- *Binding dinámico:* Los objetos podrían obtenerse desde cualquier lugar, posiblemente a través de la red. Se necesita poder enviar mensajes a objetos sin tener que conocer sus tipos específicos, a la vez que se escribe el código. El binding dinámico provee máxima flexibilidad mientras se ejecuta un programa.

A continuación, enumeramos las principales características de Java:

- *Clases*

Una clase es una construcción de software que define las variables de instancia y los métodos de un objeto. Es un patrón que define cómo aparece y se comporta un objeto, cuando es creado o instanciado desde una especificación declarada por la clase.

Ejemplo: la declaración de una clase punto será:

```
class Punto extends Object {
        public double x;        /* variable de instancia */
}
```

- *Instanciar un objeto desde su clase*

Teniendo declarado el tamaño y alcance de la clase Punto, cualquier otro objeto ahora puede crear un objeto Punto y una instancia de dicha clase como sigue:

```
Punto miPunto;              /* declara una variable referida al objeto punto
*/
miPunto = new Punto(); /* aloca una instancia de un objeto punto */
```

Se accede a las variables de este objeto punto mediante sus nombres.

```
miPunto.x = 10.0;
```

Se puede acceder de esta manera a las variables de instancia, pues fueron declaradas como públicas en la definición de clase, sino, se necesitarían métodos de acceso.

- *Constructores*

Los constructores son métodos con el mismo nombre que la clase.

```
class Punto extends Object {
        public double x,y;
        Punto(){
                x = 0.0;
        }
        Punto(double x, double y){
                this.x = x;
```

```
            this.y = y;
        }
    }
```

Cuando se crea un objeto de la clase Punto, el método constructor es usado para ejecutar cualquier inicialización necesaria, para setear las variables de instancia a un estado inicial.

Los objetos creados pueden ser inicializados por default o con valores específicos. Por ejemplo:

```
Punto inferiorIzquierdo;
Punto superiorDerecho;
inferiorIzquierdo = new Punto();                    /* inicializa */
superiorDerecho = new Punto(100.0, 200.0);          /* inicializa */
```

El constructor usado se determina por el tipo y número de parámetros en la invocación. La variable this referencia al objeto receptor. Se usa para clarificar a qué variable nos estamos refiriendo.

- *Métodos y mensajes*

En un lenguaje de programación orientado a objetos, se usan dos objetos para hacer un trabajo, el primero envía un mensaje al segundo y en respuesta, el segundo selecciona los métodos apropiados para invocar.

Usando el paradigma de pasaje de mensajes de la programación orientada a objetos se pueden construír redes de objetos que pasan mensajes entre ellos para cambiar estados.

Esta técnica de programación es la mejor forma de crear modelos y simulaciones de sistemas complejos del mundo real.

Redefinimos la clase Punto con variables privadas:

```
class Punto extends Object {
    private double x;
    Punto(){
        x = 0.0;
    }
    public void setX(double x){          /* método de acceso */
        this.x = x;
    }
}
```

Seteo de la variable x vía el método de acceso:

```
Punto miPunto;
miPunto = new Punto();
miPunto.setX(10.0);
```

Con las variables privadas se ocultan detalles de implementación internos de una clase, permitiendo cambios futuros en la implementación, sin tocar código que use la clase. Esto no ocurre con las variables públicas, donde se exponen algunos detalles de implementación de la clase.

- *Finalizadores*

Se pueden declarar finalizadores opcionales que ejecutarán acciones cuando el recolector de residuos deba liberar un objeto.

Ejemplo:

```
protected void finalize(){
    try{
```

```
                file.close();
            }catch (Exception e){;}
        }
```

En este caso en particular, el método cierra un archivo de entrada/salida que fue usado por el objeto.

- *Subclases*

La generación de subclases es el mecanismo por el cual objetos nuevos e instanciados se pueden definir en términos de objetos existentes.

Las subclases nos habilitan a usar código que ya ha sido desarrollado y, mucho más importante aún, testeado para accesos más genéricos. Se sobreescriben las partes de la clase que sean necesarias, para lograr un comportamiento específico. Al permitirse el reuso de código existente, se decrementa el tiempo de diseño, desarrollo y testeo.

Ejemplo:

```
class Punto extends Object{
        protected double x, y;
        Punto(){
                x = 0.0;
                y = 0.0;
        }
}
class TresPuntos extends Punto{
        protected double z;
        TresPuntos(){
                x = 0.0;
                y = 0.0;
                z = 0.0;
        }
}
```

La clase TresPuntos agrega la variable de instancia z y hereda las variables de instancia x e y de la clase Punto original.

- *Control de acceso*

Cuando se declara una nueva clase en Java, se puede indicar el nivel de acceso permitido para las variables de instancia y los métodos. Java provee cuatro niveles de acceso. Tres de ellos deben especificarse explícitamente, son: público, protegido y privado. El cuarto no tiene nombre, a menudo es llamado "amigable", y es el nivel de acceso que se obtiene si no se especifica otro. Este indica que las variables y métodos son accesibles por todos los objetos que estén en el mismo package, pero son inaccesibles por objetos fuera del package.

El nivel de acceso "amigable" comienza si creamos packages de clases que están relacionadas a otras. Cada una de ellas puede acceder a las variables de instancia de las otras. Los package son constructores del lenguaje Java que acumulan colecciones de clases relacionadas en una unidad simple. El beneficio secundario, desde el punto de vista del programador, es que las variables de instancia y los métodos "amigables" están disponibles para todas las clases del mismo package, pero no para clases definidas fuera de él.

Los métodos y variables de instancia públicos están disponibles para cualquier otra clase de cualquier lugar.

Protegido significa que los métodos y variables de instancia diseñados son accesibles sólo por la subclase de la clase.

Los métodos y variables de instancia privados son accesibles sólo en la clase en la cual son declarados, no están disponibles para sus subclases.

- *Variables y métodos de clase*

Una variable de clase es local a su propia clase, hay una única copia de ella y es compartida por cada objeto que se instancie en la clase. Las variables y métodos de clase son declarados como estáticos.

Ejemplo:

```
class Rectángulo extends Object {
        static final int version = 2;
}
```

Cada instancia de Rectángulo, compartirá esta variable.

Los métodos de clase son comunes a toda la clase. Se deberían usar para definir un comportamiento que sea común a cada objeto de la clase.

Los métodos de clase pueden operar solamente sobre variables de clase. No pueden acceder a variables de instancia, ni pueden invocar a métodos de instancia.

- *Métodos abstractos*

Los métodos abstractos son un constructor poderoso en el paradigma orientado a objetos. Para entenderlos, veamos la noción de superclase abstracta. Una superclase abstracta es una clase que define métodos que no son implementados por esa clase, ella sólo provee una ubicación tal que las subclases subsiguientes deban sobreescribir estos métodos y suplantar sus implementaciones actuales.

La clase abstracta define un estado y un comportamiento genérico, pero nunca veremos una implementación real de una clase abstracta, lo que se ve es una subclase concreta de la clase abstracta.

8.2.3.- Arquitectura Neutral

La solución que adopta Java para resolver el problema de distribución binaria es el *formato de código binario* que es independiente de las arquitecturas de hardware, de las interfaces del sistema y de los sistemas de ventanas. El formato de este código binario independiente del sistema es de arquitectura neutral. Si el sistema de runtime de Java está disponible en determinada plataforma de software y hardware, entonces una aplicación escrita en Java puede ejecutarse en ella, sin necesidad de ningún trabajo de puertos de comunicación para dicha aplicación.

El compilador Java no genera códigos de máquina en el sentido de instrucciones de hardware natural, sino que genera bytecodes: código de máquina independiente y de alto nivel para una máquina hipotética implementada por el intérprete Java y el sistema de runtime.

Los bytecodes de Java fueron diseñados para ser fáciles de interpretar en cualquier máquina o para ser traducidos dinámicamente a código de máquina natural, si fuera necesario por demandas de performance.

8.2.4.- Portable

El principal beneficio de los bytecodes interpretados es que un programa Java compilado puede ser portado a cualquier sistema en el que se hayan implementado el intérprete de Java y el sistema de runtime.

Tanto C como C++ tienen el problema de que muchos de sus tipos de datos fundamentales son dependientes de la implementación. Los programadores trabajan para asegurar que los programas sean portables a otras arquitecturas programando hacia un común denominador menor.

Java elimina esto definiendo un comportamiento standard que podrá ser aplicado a los tipos de datos en todas las plataformas. Los tipos de datos son:

- byte de 8 bits complementado a 2
- short de 16 bits complementado a 2
- int de 32 bits complementado a 2
- long de 64 bits complementado a 2
- float de 32 bits IEEE 754 punto flotante
- double de 64 bits IEEE 754 punto flotante
- char de 16 bits Unicode character

Fueron elegidos porque son entendidos por los microprocesadores de todas las arquitecturas. El ambiente de Java es portable a nuevas arquitecturas y sistemas operativos. El compilador Java está escrito en Java.

8.2.5.- Robusto

Java fue creado con la intención de que sirva para desarrollar software robusto, altamente confiable y seguro, de diversas formas. Hay mucho énfasis en los chequeos tempranos de posibles problemas, así como también en chequeos dinámicos posteriores para eliminar errores.

El compilador Java realiza chequeos en tiempo de compilación para poder detectar errores relacionados con la sintaxis. Una de las ventajas de un lenguaje fuertemente tipado es que permite chequeos en tiempo de compilación. Por ello Java requiere de declaraciones de tipos explícitas y no soporta declaraciones implícitas, al estilo de C.

Muchos de los chequeos en tiempo de compilación son llevados también a chequeos en tiempo de ejecución, para verificar la consistencia y para proveer mayor flexibilidad. El linkeador entiende el sistema de tipos y repite muchos de los chequeos hechos por el compilador para protección de errores de versiones.

8.2.6.- Interpretado y Dinámico

El compilador Java genera bytecodes para la máquina virtual Java, para la cual debe estar disponible un intérprete para cada arquitectura de hardware y sistema operativo sobre el que se corren las aplicaciones del lenguaje Java. Después de tener el

intérprete del lenguaje Java y un soporte en tiempo de ejecución disponible sobre una plataforma del sistema operativo y un hardware dado, se puede correr cualquier aplicación en dicho lenguaje, desde cualquier lugar, asumiendo siempre que esas aplicaciones están escritas en forma portable. No existe la noción de una fase separada de linkeo, después de la compilación, en el ambiente Java.

El lenguaje Java, por ser portable e interpretado, produce sistemas altamente dinámicos y dinámicamente extensibles. Las clases son linkeadas según sean requeridas, y pueden cargarse desde la red. El código que llega es verificado antes de ser pasado al intérprete para su ejecución.

La programación orientada a objetos fue aceptada para resolver problemas de software, por usar encapsulamiento de datos y procedimientos, y reuso de código. Uno de los problemas que soluciona es el de la superclase frágil, explicado en el Apéndice A.

8.2.7.- Seguridad [27]

El lenguaje Java permite a los browsers compatibles con Java traer fragmentos de código dinámicamente y luego ejecutarlos localmente. Sin embargo, los usuarios deben tener cuidado con el código que ejecutan porque podría venir de fuentes no confiables o podría haber pasado por redes inseguras.

Desde el momento en que el código compilado del lenguaje Java está diseñado para ser transportado en formato binario a través de las redes, la seguridad es sumamente importante.

Debido a que los bytecodes de Java corren en el host, hay ciertos requerimientos de seguridad. Los usuarios que bajan archivos de clase de Java de lugares remotos y, posiblemente, inseguros, deben saber que el código traído no puede destruír al intérprete de bytecodes de Java para realizar operaciones no permitidas.

Los niveles más bajos del intérprete de Java implementan la seguridad de distintas formas:

- *Seguridad por estar publicado*

  Los códigos fuente completos para el intérprete y el compilador de Java están disponibles para su inspección.

- *Seguridad por estar bien definido*

  El lenguaje Java es estricto en su definición:

  - Se garantiza que todos los tipos primitivos en el lenguaje sean de un tipo específico.
  - Todas las operaciones están definidas para que se realicen en un orden específico.

- *Seguridad gracias a la falta de punteros aritméticos*

  El lenguaje Java no tiene punteros aritméticos, así que los programadores Java no pueden crear un puntero a memoria. Todas las referencias a métodos y variables de instancia en los archivos de clase son vía nombres simbólicos.

- *Seguridad por el recolector de residuos*

La recolección de residuos hace que los programas Java sean más seguros y robustos.

Una falla en la liberación de memoria puede causar que un programa use más cantidad de ella. Accidentalmente, liberar dos veces la misma parte de memoria puede causar problemas que son difíciles de localizar. El lenguaje Java elimina la necesidad de que los programadores tengan que manejar estos inconvenientes.

- *Seguridad gracias a un chequeo estricto en tiempo de compilación*

El compilador Java realiza chequeos en tiempo de compilación que son largos y estrictos de manera que se puedan detectar la mayor cantidad de errores posible. El lenguaje Java es fuertemente tipado, el sistema de tipos no tiene roturas:

- A los objetos no se les puede hacer un cast sin un chequeo explícito en tiempo de ejecución.
- Todas las referencias a métodos y variables son chequeadas para asegurar que los objetos sean del tipo apropiado.
- Los enteros no pueden convertirse en objetos. Los objetos no pueden ser convertidos en enteros.

El compilador también asegura estrictamente que un programa no acceda a valores de una variable local no inicializada.

Todos los archivos de clase traídos desde afuera están sujetos a un verificador. Este asegura que un archivo tenga el formato correcto. El verificador de bytecodes también mejora la performance del intérprete, y es independiente del compilador de Java.

El tema de seguridad es tratado en más detalle, debido a su importancia, en el Apéndice A.

8.2.8.- Multithreading

Los threads son una clave esencial de Java. Una librería de Java provee la clase Thread que soporta una colección rica de métodos para comenzar, correr, parar y chequear el estado de un thread.

El soporte de threads de Java incluye un conjunto sofisticado de primitivas de sincronización basadas en el uso del paradigma de monitor y variable condición. El soporte de la integración de threads en el lenguaje, hace que ellos sean mucho más fáciles de usar y más robustos.

Los threads Java son preventivos, y dependen de la plataforma en la cual se ejecuta el intérprete Java; los threads también pueden realizarse en tiempos discontinuos. En sistemas que no soportan la ejecución discontinua, cuando un thread comienza, la única manera de abandonar el control del proceso es si otro thread de mayor prioridad toma el control del mismo. En aplicaciones de computación intensiva se debe considerar cómo cortar el control periódicamente usando el método yield() para darle la oportunidad de correr a otros threads; haciendo ésto, se asegurará una mejor respuesta interactiva para aplicaciones gráficas.

Java soporta multithreading a nivel del lenguaje y vía soporte de su sistema de tiempo de ejecución y objeto thread. A nivel del lenguaje, los métodos que son declarados sincronizados en una clase no corren concurrentemente. Tales métodos corren bajo el control de monitores para asegurar que las variables permanezcan en un estado consistente. Cada clase y objeto instanciado tienen su propio monitor, que entra en ejecución si se lo requiere.

Cuando se entra a un método sincronizado, éste adquiere un monitor sobre el objeto actual. El monitor excluye cualquier otro método sincronizado en ese objeto desde la ejecución. Cuando el método sincronizado retorna, se libera su monitor. Ahora, otro método sincronizado del mismo objeto está libre para correr.

Si se quiere que los objetos sean threads seguros, cualquier método que deba cambiar los valores de las variables de instancia debería ser declarado sincronizado. Esto asegura que un único método puede cambiar el estado de un objeto a la vez. Los monitores de Java son re-entrantes: un método puede adquirir el mismo monitor más de una vez.

8.2.9.- Perfomance

Java ha sido portado y corrido sobre distintas plataformas de hardware ejecutando sobre diferentes sistemas operativos.

Los números que reflejan la performance de los bytecodes interpretados son por lo general lo suficientemente adecuados para correr aplicaciones interactivas gráficas; podrían aparecer complicaciones cuando se requiera una mayor performance. En tales casos, los bytecodes de Java pueden ser traducidos en tiempo de ejecución a código de máquina, para una determinada CPU sobre la cual se está ejecutando la aplicación. La performance de los bytecodes convertida a código de máquina es casi igual a la de C o C++.

8.3.- RMI (Remote Method Invocation) [30]

8.3.1.- Introducción

Los sistemas distribuidos requieren que las computaciones que corren sobre diferentes espacios de direcciones, potencialmente sobre diferentes hosts, sean capaces de comunicarse. Como mecanismo de comunicación básico, Java soporta sockets, los cuales son flexibles y suficientes para la comunicación en general. Sin embargo, los sockets requieren que los clientes y los servers estén relacionados en protocolos a nivel de aplicación, para codificar y decodificar mensajes para el intercambio; el diseño de tales protocolos es tedioso y propenso a errores.

Una alternativa a los sockets es la llamada a procedimientos remotos (RPC), lo cual abstrae la interface de comunicación a un nivel de llamadas a procedimientos. En lugar de trabajar directamente con sockets, el programador tiene la ilusión de llamar a un procedimiento local, cuando en realidad los argumentos de la llamada se empaquetan y envían al destino remoto de la misma.

RPC, sin embargo, no se adapta bien a sistemas de objetos distribuidos, donde se necesita comunicación entre objetos a nivel de programas que residen en espacios de direcciones diferentes. Para matchear con la semántica de la invocación a objetos, los sistemas de objetos distribuidos requieren de la invocación de métodos remotos o RMI. En tales sistemas, un objeto representante local (stub) maneja la invocación sobre el objeto remoto.

El sistema de invocación de métodos remotos de Java ha sido específicamente diseñado para operar en el ambiente Java. Si bien otros sistemas RMI pueden adaptarse para manejar objetos Java, ellos no tienen una integración directa con el sistema Java, debido a sus requerimientos de interoperabilidad con otros lenguajes.

8.3.2.- Objetivos del Sistema

Los objetivos para el soporte de objetos distribuidos en el lenguaje Java son:

- Soporte de invocación remota sobre objetos que están en diferentes máquinas virtuales.
- Soporte de respuestas desde los servers a las applets.
- Integración del modelo de objetos distribuidos en el lenguaje Java de manera natural, manteniendo la mayoría de su semántica de objetos.
- Hacer diferencias entre el modelo de objetos distribuido y el modelo de objetos local aparente.
- Permitir la escritura de aplicaciones distribuidas confiables de la manera más simple posible.
- Preservar la seguridad provista por el sistema de tiempo de ejecución de Java.

Por debajo de todos estos objetivos existe un requerimiento general que es que el modelo RMI sea simple (fácil de usar) y natural (se adapte bien al lenguaje).

8.3.3.- Modelo de Objetos Distribuido Java

En el modelo de objetos distribuido Java, un objeto remoto es aquel cuyos métodos pueden invocarse desde otra máquina virtual Java, potencialmente sobre un host diferente. Un objeto de este tipo está descripto por una o más interfaces remotas, las cuales son interfaces Java que declaran los métodos de un objeto remoto.

La invocación de métodos remotos (RMI) es la acción de invocar un método de una interface remota sobre un objeto remoto. Más importante aún, la invocación a un método sobre un objeto remoto tiene la misma sintaxis que la invocación a un método sobre un objeto local.

8.3.3.1.- Contraste entre los modelos distribuidos y no distribuidos

El modelo de objetos distribuido Java es similar al modelo de objetos Java en lo siguiente:

- Una referencia a un objeto remoto puede pasarse como argumento o retornarse como resultado de cualquier invocación a un método (local o remoto).

- A un objeto remoto se le puede hacer un cast a cualquier conjunto de interfaces remotas soportadas por la implementación, usando la sintaxis Java predefinida.

- Se puede usar el operador predefinido de Java instanceOf para testear las interfaces remotas sosportadas por un objeto remoto.

El modelo de objetos distribuido Java difiere del modelo de objetos Java en lo siguiente:

- Los clientes de objetos remotos interactúan con interfaces remotas, nunca con las clases que implementan esas interfaces.

- Los argumentos no remotos y los resultados de una invocación de métodos remotos son pasados por copia. Esto es porque las referencias a objetos son únicamente útiles dentro de una misma máquina virtual.

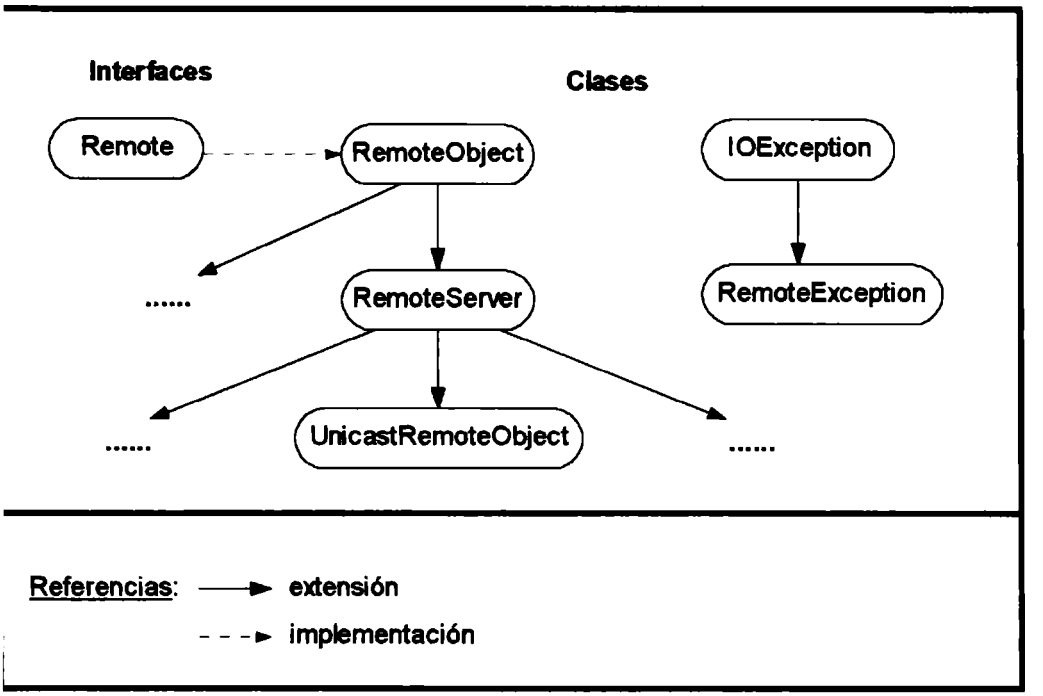
- Un objeto remoto se pasa por referencia, no a través de la copia de la implementación remota actual.

- La semántica de algunos de los métodos definidos por la clase Object está especializada para objetos remotos.

- Debido a que los modos de falla de la invocación de objetos remotos es inherentemente más complicada que los modos de falla de la invocación de objetos locales, los clientes deben manejar excepciones adicionales que pueden ocurrir durante dicha invocación.

8.3.3.2.- Clases e interfaces RMI

Las interfaces y clases que son responsables de la especificación del comportamiento remoto del sistema RMI están definidas en los paquetes java.rmi y java.rmi.server. La siguiente figura muestra la relación entre estas interfaces y clases.



?.1.- La interface Remote

Todas las interfaces remotas extienden, ya sea directa o indirectamente, a ce java.rmi.Remote, la que no define métodos.

Los métodos en una interface remota deben definirse como siguen:

- Cada método debe declarar java.rmi.RemoteException en su cláusula throv además de las excepciones específicas de la aplicación.
- Un objeto remoto pasado como argumento o valor de retorno (ya s directamente o embebido en un objeto local) debe estar declarado como interface remota y no como la clase de la implementación.

?.2.- La clase RemoteException

La clase java.rmi.RemoteException es la superclase de todas las excepciones q tirar el sistema de runtime RMI. Para asegurar la robustez de las aplicaciones q sistema RMI, cada método declarado en una interface remota debe especifi ni.RemoteException en su cláusula throws.

java.rmi.RemoteException es tirada cuando falla la invocación a un méto o (por ejemplo cuando falla la red o no se puede acceder al server de la llamad ermite que la aplicación haga la invocación remota para determinar cómo mane la excepción remota.

?.3.- La clase RemoteObject y sus subclases

Las funciones del server RMI están provistas por la cla mi.server.RemoteObject y sus subclases, java.rmi.server.RemoteServer mi.server.UnicastRemoteObject:

- La clase java.rmi.server.RemoteObject provee la semántica remota de Object implementando los métodos hashCode, equals y toString.

- Las funciones necesarias para crear objetos y exportarlos (hacerlos disponibles remotamente) están provistas, abstractamente, por java.rmi.server.RemoteServer, y concretamente, por sus subclases. Las subclases identifican la semántica de la referencia remota, por ejemplo, si el server es un objeto simple o replicado (requiriendo comunicaciones con múltiples ubicaciones).

- La clase java.rmi.server.UnicastRemoteObject define un objeto remoto cuyas referencias son válidas únicamente mientras está vivo el proceso server.

8.3.3.3.- Implementación de una interface remota

Las reglas generales para una clase que implementa una interface remota son las siguientes:

- La clase generalmente extiende a java.rmi.server.UnicastRemoteObject, heredando así el comportamiento remoto provisto por las clases java.rmi.server.RemoteObject y java.rmi.server.RemoteServer.

- La clase puede implementar cualquier número de interfaces remotas.

- La clase puede extender a cualquier otra clase de implementación remota.

- La clase puede definir métodos que no aparezcan en la interface remota, pero esos métodos sólo pueden ser usados localmente, o sea no estarán disponibles remotamente.

8.3.3.4.- Equivalencia de tipos de objetos remotos con stubs locales

En el modelo de objetos distribuido, los clientes interactúan con objetos stub (representantes) que tienen exactamente el mismo conjunto de interfaces remotas que las definidas por la clase del objeto remoto; la clase del stub no incluye las porciones no remotas de la jerarquía de clases que constituye el grafo de tipos del objeto. Esto es porque la clase del stub está generada a partir de la implementación de la clase más refinada que implementa una o más interfaces remotas. Por ejemplo, si C extiende a B, y B extiende a A, pero sólo B implementa una interface remota, entonces se genera un stub a partir de B y no de C.

Debido a que el stub implementa el mismo conjunto de interfaces remotas que la clase del objeto remoto, el stub tiene, desde el punto de vista del sistema Java, el mismo tipo que las porciones remotas del grafo de tipos del objeto server. Además, un cliente puede hacer uso de las operaciones predefinidas de Java para chequear el tipo de un objeto remoto y para hacer un cast de una interface remota a otra.

8.3.3.5.- Pasaje de parámetros en RMI

Un argumento o valor de retorno de un objeto remoto puede ser cualquier tipo Java que sea serializable. Esto incluye tipos primitivos, objetos remotos y objetos no remotos que implementen la interface java.io.Serializable. Para las applets, si la clase de un argumento o valor de retorno no está disponible localmente, ésta es cargada dinámicamente vía el AppletClassLoader. Para las aplicaciones, estas clases son cargadas por el cargador de clases que cargó la aplicación, ya sea el cargador de clases por defecto (el cual usa el classpath local) o el RMIClassLoader (el cual usa el codebase del server).

Algunas clases, por no ser serializables, no se pueden pasar, por razones de seguridad. En este caso la invocación de métodos remotos fallará con una excepción.

8.3.3.5.1.- Pasaje de objetos no remotos

Un objeto no remoto pasado como parámetro o retornado como resultado de una invocación de métodos remotos es pasado por copia.

Cuando aparece un objeto no remoto en una invocación de métodos remotos, su contenido es copiado antes de invocar la llamanda sobre el objeto remoto. De la misma manera, cuando se retorna un objeto remoto desde una invocación de métodos remotos, se crea un nuevo objeto en la máquina virtual llamante.

8.3.3.5.2.- Pasaje de objetos remotos

Cuando se pasa como parámetro un objeto remoto, se esta pasando el stub del mismo. Un objeto remoto pasado como parámetro sólo puede implementar interfaces remotas.

8.3.3.6.- Manejo de excepciones en RMI

Debido a que los métodos remotos incluyen a java.rmi.RemoteException en su signatura, el llamante debe estar preparado para manejar estas excepciones además de las específicas de la aplicación. Cuando se tira una excepción java.rmi.RemoteException durante una invocación de métodos remotos, el cliente puede tener algo o nada de la información con respecto al resultado de la llamada -si la falla ocurrió antes, durante o después de que dicha llamada se completara. Además, las interfaces remotas y los métodos llamantes declarados en ellas deberían diseñarse con esta semántica de fallas en mente.

8.3.3.7.- Sobreescritura de métodos por RMI

La implementación por defecto de los métodos equals, hashCode y toString en la clase Object no es apropiada para objetos remotos. Por esto, la clase java.rmi.server.RemoteObject provee implementaciones para estos métodos con una semántica más apropiada para los objetos remotos. De esta forma, todos los objetos que necesiten estar disponibles remotamente pueden extender a la clase java.rmi.server.RemoteObject (por lo general indirectamente vía java.rmi.server.UnicastRemoteObject).

8.3.3.8.- Semántica de métodos de Object declarados como final

Los siguientes métodos están declarados como final en la clase Object y no puueden sobreescribirse:

- getClass
- notify
- notifyAll
- wait

8.3.3.9.- Localización de objetos remotos

El modelo de objetos distribuido Java provee un server simple de nombres de arranque para el almacenamiento de referencias a objetos remotos. Una referencia a un objeto remoto puede almacenarse usando los métodos de la clase java.rmi.Naming basados en URL.

Un cliente, para poder invocar un método sobre un objeto remoto, debe primero obtener la referencia al objeto. Una referencia a un objeto remoto generalmente se obtiene como valor de retorno de una llamada a un método. El sistema RMI provee un servidor de nombres de arranque simple del cual se puede obtener los objetos remotos de un host dado. La clase java.rmi.Naming provee métodos basados en URL (Uniform Resourse Locator) para buscar, enlazar, reenlazar, desenlazar y listar los pares nombre-objeto guardados en un host y port particular.

Sección II

8.4.- AMCO: Visión Interna

El ambiente AMCO está basado en el modelo de objetos distribuidos, explicado en la sección 8.1, que resultó de la corvergencia de las tecnologías orientada a objetos y cliente/servidor. Nuestra aplicación tiene una estructura cliente/servidor, ya que existen un cliente y tres servers. Además, utiliza un paradigma orientado a objetos a efectos de facilitarse el desarrollo e implementación, aprovechando sus características de encapsulamiento, polimorfismo, herencia y binding dinámico. El lenguaje Java, detallado en la sección 8.2 y en el Apéndice A, es utilizado para la implementación tanto de los clientes como de los servers; dicho lenguaje es quien nos brinda esta tecnología orientada a objetos. El paquete adicional RMI de Java, al que se hace referencia en la sección 8.3 y en el Apéndice B, es el que provee la comunicación entre los objetos que residen en espacios de direcciones diferentes a nivel del programa.

8.4.1- Requerimientos

Un usuario que desee utilizar AMCO necesita disponer de:

- Netscape 3.0.1.
- Extensión de RMI para Netscape 3.0.1, disponible a través del archivo "rmi-netscape.zip".

- Algún editor HTML para crear/modificar las páginas a compartir.
- Una aplicación FTP, para las transferencias de páginas desde y hacia la máquina host.

8.4.2.- Lenguaje de implementación

AMCO está totalmente implementado en Java Development Kit (JDK) 1.0.2 -sección 8.2 y en el Apéndice A-, extendido con el paquete RMI (Remote Method Invocation) -sección 8.3 y Apéndice B-, que es el que permite la distribución usando comunicación entre objetos, que residen en espacios de direcciones diferentes, a nivel de programa.

Esta versión de Java es la soportada por la mayoría de los browsers WWW. En particular, es la versión soportada por Netscape 3.0.1, que es el único browser en el que se puede ejecutar AMCO. Esta restricción se debe a que el paquete RMI sólo está disponible para dicho browser, y siempre que se haya ejecutado, en primer lugar, la extensión de RMI para Netscape, mencionada en la sección anterior.

8.4.3.- Arquitectura

El AMbiente COlaborativo AMCO está arquitecturalmente construído a partir del modelo de objetos distribuido (sección 8.1). Dicha arquitectura usa la tecnología Internet (capítulo 4), como plataforma de base. Como lo dijimos en la sección previa, cada componente de AMCO está totalmente implementado en Java, y el paquete adicional RMI, los que aportaron los inmejorables beneficios de la orientación a objetos y la posibilidad de invocación a métodos remotos, con la misma sintaxis que la invocación a métodos sobre un objeto local. Destacamos estas características por ser la base del modelo utilizado, pero sin menospreciar al resto de las capacidades del lenguaje Java, que también utilizamos en esta implementación. Dichas capacidades, como la de ser un lenguaje interpretado, dinámico, multithreading, portable, de alta performance, entre otras, son expresadas detalladamente en la sección 8.2.

AMCO tiene una estructura basada en un modelo cliente/servidor, que comprende:

1. Tres servers:

    - Un server del ambiente general (ServerInterf), que maneja toda la información referente a los usuarios conectados.

    - Un server del chat (ServerChat), que se encarga del manejo y distribución de los mensajes entre los usuarios conectados, y

    - Un server del Manejador de Páginas (ServerGenPag), que tiene como tarea administrar toda la información referente a las páginas a desarrollar, autores de las mismas y a todo el sistema de votaciones relacionado.

2. Una aplicación cliente.

Los servers recientemente mencionados deben ser activados antes de que la primera aplicación cliente comience a correr. Cuando los usuarios se conectan al sistema y realizan la activación de los servers, el programa internamente determina si éstos están corriendo, en ese caso la nueva activación queda sin efecto; en caso contrario, se concreta. Además de arrancar los tres servers propios de nuestra aplicación, se ejecuta también el server Registry de RMI, que es un server de nombres que permite a los clientes remotos localizar una referencia a un determinado objeto remoto. Este es usado solamente para localizar al primer objeto remoto con el que necesita dialogar la aplicación, luego ese objeto es el que se encargará de proveer a dicha aplicación del soporte específico para encontrar a los objetos restantes.

La necesidad del arranque repetitivo de los servers por parte de los usuarios del sistema, se debe a que AMCO es una demo, y como tal se usa sólo ocacionalmente. Por ello no le encontramos sentido a que dichos servers estén continuamente activados.

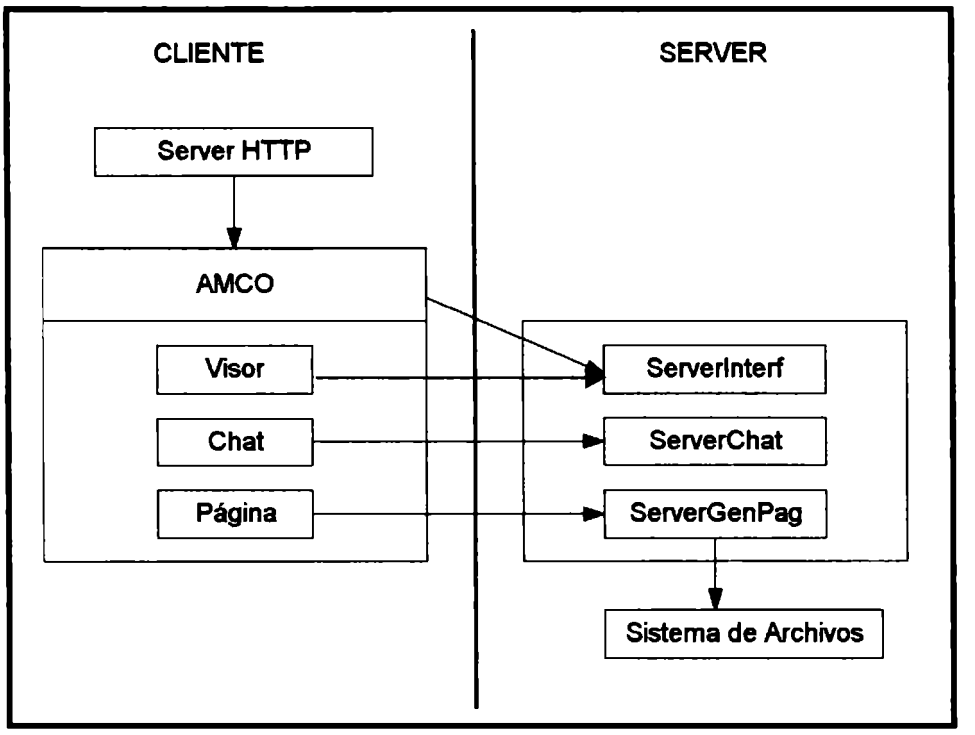
La decisión de utilizar tres servers en lugar de uno para prestar los servicios requeridos por la aplicación cliente, fue tomada por una cuestión de claridad en el diseño. Es decir, se asocia un server a cada una de las herramientas contenidas en el ambiente, así, habrá un server asociado al Chat, uno asociado al Manejador de Páginas, y uno al ambiente AMCO en general, incluyendo al visor. De esta manera, los servicios requeridos por las herramientas fueron agrupados de acuerdo a su funcionalidad.

La información que contienen el server del chat y el server del ambiente general, se maneja solamente en memoria, ya que ésta es relevante únicamente durante el tiempo en que los usuarios están conectados al sistema. No ocurre lo mismo con el server del Manejador de Páginas, pues su información es nuevamente utilizada cada vez que el usuario comienza una sesión. Por este motivo, dicho server, cada determinados intervalos de tiempo hace un backup de su estado actual en el sistema de archivos de la máquina host. Cada vez que este server sea arrancado, se recupera su último estado a partir de la información guardada.

La comunicación entre los clientes y los servers se realiza mediante una red. El sistema está conectado a un server HTTP para poder ser usado con browsers WWW. El server HTTP envía las clases Applets a los browsers que la pidan (gracias a las características del lenguaje Java: Interpretado y Dinámico, expresadas en la sección 8.2.6). Una vez cargadas las applets se conectan directamente a los servers vía un puerto TCP/IP.

Los servers soportan un protocolo de llamadas a procedimientos remotos para la comunicación cliente/servidor sobre TCP/IP. Los mensajes al estilo RPC están construidos como objetos Java usando el paquete RMI (Invocación a Métodos Remotos), que es una extensión aplicable al JDK 1.0.2. Este paquete brinda la posibilidad de invocar a métodos remotos, con la misma sintaxis que la invocación a métodos sobre objetos locales. La teoría relacionada con este tema está explicada en la sección 8.3 y en el Apéndice B.

Esquema de la comunicación cliente/servidor de AMCO



8.4.4.- Diseño

El diseño de AMCO se llevó a cabo cumpliendo una serie de pasos. El primero de ellos fue determinar una especificación general del sistema. Esta describe la forma en que debiera comportarse el ambiente, sus objetivos y las interacciones con el usuario, según lo expuesto en la sección 7.3.

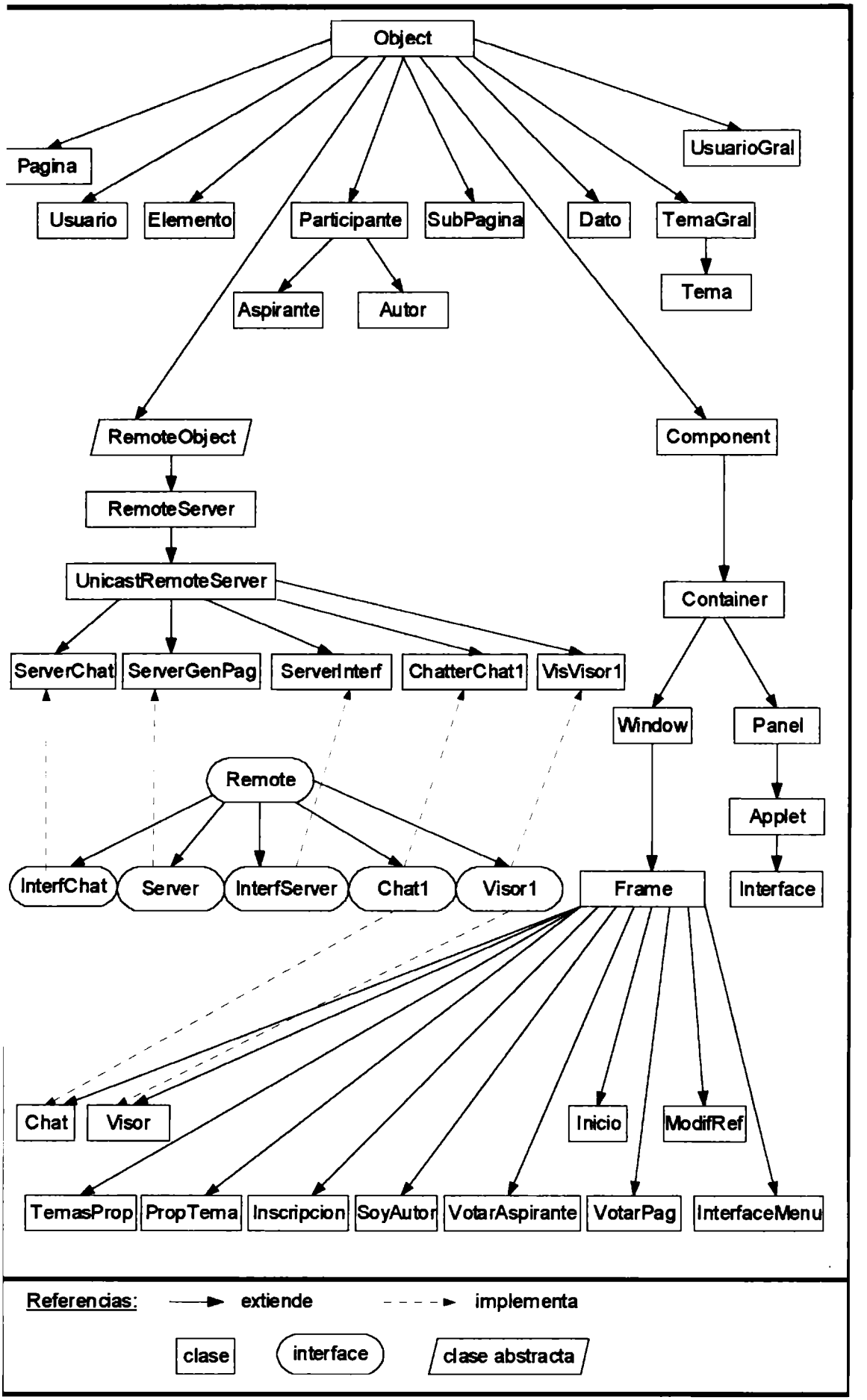
Luego se utilizó un modelo de casos de uso (Apéndice C) para comprender en profundidad el comportamiento del sistema. Así quedaron identificados los actores, que representan las necesidades de intercambio de información con el sistema, y los casos de uso, que representan la funcionalidad completa del mismo.

Posteriormente, se llevó a cabo la transformación de esta visión externa del sistema brindada por los casos de uso, a un diseño orientado a objetos (Apéndice C). Para ésto fue necesario determinar las clases candidatas. Las primeras surgieron de los actores y casos de uso encontrados. Otras se pudieron determinar a partir de los sustantivos mencionados en éstos.

Una vez que logramos así todas las clases candidatas, identificamos sus características comunes, a fin de organizar los objetos similares en una clasificación jerárquica. Luego, determinamos con qué tipo de objetos se corresponden cada una de estas clases y encontramos: objetos proveedores de servicios, que serían los servers; objetos interfaces, que serían la cara visible del sistema; y objetos controladores de información, que serían los que contienen los valores internos a ser consultados.

Una vez obtenida esta estructura de clases, determinamos las responsabilidades de cada una, y las colaboraciones entre ellas, a partir de las cuales surgieron los métodos.

El siguiente gráfico expresa la jerarquía de clases de AMCO.


Object
Pagina
Usuario
Elemento
Participante
SubPagina
Dato
TemaGral
UsuarioGral
Aspirante
Autor
Tema
RemoteObject
RemoteServer
UnicastRemoteServer
ServerChat
ServerGenPag
ServerInterf
ChatterChat1
VisVisor1
Component
Container
Window
Panel
Applet
Interface
Remote
InterfChat
Server
InterfServer
Chat1
Visor1
Frame
Chat
Visor
Inicio
ModifRef
TemasProp
PropTema
Inscripcion
SoyAutor
VotarAspirante
VotarPag
InterfaceMenu
Referencias:
extiende
implementa
clase
interface
clase abstracta

Conclusiones

Al llegar al final de nuestra tesis, podemos asegurar que hemos aprendido en detalle el tema colaboración. Nos enfocamos específicamente en las ventajas que puede aportar dicho tema a los ambientes computacionales y nos dimos cuenta que su alcance es realmente importante. Hasta el día de hoy el tema está en sus comienzos, pero su proyección a futuro es muy amplia, pues su tendencia es abarcar cada vez más áreas de trabajo, y a su vez profundizar en cada una de ellas. De ahí lo interesante del tema.

El trabajo en grupo permite la unión de las experiencias de cada uno de los integrantes, para llegar a un resultado mejor y para decrementar la cantidad de tiempo de desarrollo, pues, ningún individuo tiene toda la experiencia, información, diversidad de puntos de vista o el tiempo para llevar a cabo un proyecto.

La tendencia es lograr que desaparezca la PC como una herramienta de trabajo individual y que la red sea la computadora. En este momento, ésto es tan imaginable como posible, debido al avance de la tecnología en general, y por sobre todo, al gran auge de Internet.

A principios del año '96, comienzo de nuestra tesis, todo lo que hablaba de colaboración terminaba haciendo referencia al lenguaje Java como la herramienta capaz de llevar a cabo aplicaciones sobre Internet en forma muy simple y eficiente, permitiendo lograr además, desarrollos interactivos, los que no eran posibles hasta el momento.

Dicho lenguaje crea un término medio atractivo entre los lenguajes de alto nivel, portables pero lentos, como los de scripting, y los lenguajes de bajo nivel, rápidos pero no portables, como los compilados. Además, por sus características de: simple, familiar, orientado a objetos, de arquitectura neutral, portable, robusto, interpretado, dinámico, seguro y multithreaded, provee un buen nivel de performance para la mayoría de las aplicaciones intensivas.

Java maneja la parte de comunicación a nivel de sockets, los que si bien son flexibles y suficientes para las comunicaciones en general, requieren que los clientes y los servers estén relacionados a nivel de aplicación, a fin de codificar y decodificar mensajes para el intercambio, lo que lleva a un diseño de protocolos tedioso y propenso a errores.

Debido a ésto, decidimos usar el paquete adicional de Java, RMI (Remote Method Invocation), que abstrae la interface de comunicación a un nivel de llamadas a procedimientos. En lugar de trabajar con sockets, el programa tiene la ilusión de llamar a un procedimiento local, cuando en realidad los argumentos de la llamada se empaquetan y envían al destino remoto de la misma.

Con este entorno, logramos desarrollar una aplicación en Java, a la que llamamos AMCO: Una Herramienta para Colaborar, la cual corre sobre Internet y utiliza los principios de la colaboración. Así, demostramos que todo lo expuesto es posible, y que con tiempo y tecnología hasta los objetivos más inverosímiles se pueden lograr.

AMCO es un AMbiente COlaborativo que logra la colaboración al permitir a los diversos integrantes de un grupo poder trabajar en conjunto para el manejo de páginas HTML asociadas a un tema determinado, de manera tal que la página resultante pueda verse como producida por una única mente.

Además, podemos asegurar que AMCO tiene múltiples extensiones posibles desde los puntos de vista teórico, técnico y de implementación.

Desde el punto de vista teórico, AMCO podría ser la base de otras herramientas colaborativas, entre ellas:

- Publicaciones de distinta índole, redactadas colaborativamente entre varios autores que pueden encontrarse en lugares dispersos geográficamente.

- Cuentos, creados por chicos de diferentes escuelas que, conectados a una red, colaboran en la generación de las páginas que los componen.

- Evaluaciones, realizadas por ejemplo, por docentes de distintas escuelas para aplicarlas a los alumnos, a fin de obtener ciertos resultados, como podrían ser valores estadísticos entre grupos de alumnos.

- Etc.

Desde el punto de vista técnico, está la posibilidad de integrar en este mismo ambiente otras herramientas colaborativas como: mail electrónico, pizarrón compartido, video conferencia, comunicación telefónica, etc.

Desde el punto de vista de la implementación, ésta se podría optimizar con el uso de bases de datos tanto en el caso de los backups del server, como para el almacenamiento de las páginas compartidas. Por otra parte, se podrían hacer mejoras a la interface, con el único motivo de lograr una presentación visiblemente más agradable. En Java, ésto no es posible, ya que el uso de imágenes sobre componentes visuales no está permitido y su simulación es muy ineficiente.

La principal dificultad que encontramos en el desarrollo de la tesis, fue el aprendizaje del lenguaje Java. Esto se debió a que comenzamos a estudiarlo cuando el lenguaje estaba en pleno desarrollo, por ello se sucedían las distintas versiones con mucha frecuencia, y no existía la bibliografía suficiente. En el momento en que se podía disponer de bibliografía referente a una determinada versión, aparecía una nueva solucionando los problemas de la anterior, por lo que era imprescindible el traslado, quedándonos otra vez sin la bibliografía adecuada. La última versión de Java es el JDK 1.1.3 que no fue utilizada pues recién al final de nuestra tesis hizo su aparición en el mercado el browser Netscape 4.0, que es el único compatible.

A principios del '96, cuando nos planteamos esta tesis, apenas comenzaba el tema colaboración, y nuestra demo aparecía como un proyecto importante, ya que en el mercado se contaba con muy pocas herramientas de este estilo. Hoy sabemos que en el mundo hay muchas aplicaciones colaborativas similares a la nuestra, pero fue una carrera contra el tiempo. Finalmente, llegamos a desarrollar un tema que está en la mira de todo el mundo y que, por otra parte, satisface nuestros objetivos iniciales.

Apéndice A

Java

1.- Historia [24]

Java surgió como un lenguaje de programación para software de productos electrónicos, como tostadoras, agendas, etc. Este tipo de software tiene requerimientos de diseño únicos, necesita ser capaz de funcionar en diferentes chips de computadoras a medida que estos evolucionan, y debe ser extremadamente confiable porque cuando un producto falla, el fabricante tiene que reemplazar todo el equipo.

Un pequeño grupo de Sun, liderado por James Gosling, que estaba trabajando en este problema, descubrió que los lenguajes de programación existentes, como C y C++, no satisfacían los requerimientos.

En consecuencia, en 1990, Gosling comenzó a diseñar un nuevo lenguaje de programación que sería más apropiado para los productos electrónicos. Este lenguaje, originalmente conocido como OAK, fue pequeño, confiable e independiente de la arquitectura. En 1993, aparece el WWW sobre Internet y se desata la tormenta. Los desarrolladores de Java se dieron cuenta de que un lenguaje de arquitectura neutral como Java sería ideal para programar sobre Internet, ya que un programa podría correr sobre todos los diferentes tipos de computadoras conectadas a dicha red.

En este punto, el desarrollo de Java tomó una nueva importancia para Sun. El equipo escribió un browser de Web, llamado HotJava, que fue el primer browser de Web que soportó las Applets de Java. Un Applet es un pequeño programa Java que puede ser embebido en otra aplicación, por ejemplo un documento HTML, para proveer de contenido ejecutable e interactivo a una página de WWW.

Cada vez más personas se interesaron en Java, lo que se incrementó aún más con la aparición del Netscape Navigator 2.0 que soportaba Applets Java. Con el apoyo de empresas como: IBM, SGI y ORACLE se puede asegurar que cada vez más productos de hardware y software se irán incorporando a la tecnología de Java.

2.- Diferencias entre Java y C/C++ [25]

- *Tipos de datos primitivos:*

En Java todo es un objeto. Hay sólo tres grupos de tipos de datos primitivos: numéricos, booleanos y arreglos.

El tipo de dato numérico Integer tiene varios tipos que son: byte de 8 bits, short de 16 bits, int de 32 bits y long de 64 bits. El tipo byte de 8 bits reemplazó al tipo char de C y C++. Java le da otra interpretación al tipo char. No hay tipos de datos sin signo para los Integer en Java.

El tipo de dato numérico Real tiene dos tipos: float de 32 bits y double de 64 bits. El tipo Real y sus operaciones aritméticas están definidos por la especificación IEEE 754.

El tipo de dato Character es una desviación del de C. Define un caracter de 16 bits Unicode. Estos son valores sin signo de 16 bits que definen códigos de caracteres que van desde 0 hasta 65.535. Por adoptar el conjunto standard de caracteres Unicode, las aplicaciones Java son fácilmente internacionalizables y localizables.

El tipo de dato Boolean fue agregado como un tipo primitivo, necesidad que surgió a partir de la práctica en C, de definir palabras claves como TRUE y FALSE o YES y NO, etc. Una variable booleana en Java puede tomar los valores true o false. Un tipo booleano en Java no puede ser convertido a un tipo numérico, como en C.

- *Operadores aritméticos y relacionales:*

En contraste con C y C++, los arreglos en Java son objetos. Se pueden declarar y alocar arreglos de cualquier tipo, como así también, arreglos de arreglos para obtener arreglos multidimensionales.

- *Strings*:

Los strings son objetos Java y no pseudo-arreglos como en C. Hay dos clases: la clase String, que es para objetos de sólo lectura (objetos inmutables) y la clase StringBuffer, que es para objetos que se desean modificar (objetos mutables).

Los strings deben indicarse entre comillas dobles : ".

Java extendió el significado del operador + para indicar la concatenación de strings.

- *Cortes multi-nivel*:

Java no tiene sentencia goto. Para cortar o continuar lazos anidados se pueden poner labels en el lazo y cortar o continuar en el bloque nombrado por el label.

El uso de bloques con labels produjo una gran simplificación de la programación y una reducción del mantenimiento.

- *Manejo de memoria y recolección de residuos*

C y C++ tienen el problema del manejo de memoria: alocarla, liberarla y seguir la pista de qué parte de ella puede ser liberada en qué momento. El manejo explícito de memoria ha probado ser una fuente fructífera de errores, crashes, falta de memoria y mala performance.

Java eliminó por completo el manejo de memoria explícito. Ya no existen los punteros al estilo C, ni los punteros aritméticos, ni el malloc, ni el free. Hay un recolector de residuos en el sistema de runtime. Una vez que fue alocado un objeto, el sistema de runtime guarda la pista del estado de los objetos y automáticamente reclama memoria cuando dejan de estar en uso, liberándola para usos futuros.

El modelo de manejo de memoria está basado en objetos y referencias a ellos. Todas las referencias a almacenamientos alocados, son a través de manejadores simbólicos. Cuando un objeto no tiene más referencias es candidato para el recolector de residuos.

- *El background del recolector de residuos*

El uso típico de aplicaciones interactivas tiene muchas pausas naturales: mientras el usuario está contemplando la escena que tiene enfrente o mientras piensa lo próximo que va a hacer. El sistema de runtime de Java aprovecha estos períodos de inactividad para correr el recolector de residuos. Este recoge y compacta la memoria no usada,

aumentando la probabilidad de tener memoria disponible, cuando haga falta, en períodos de gran uso interactivo.

- *Sincronización de threads integrados*

Java soporta multithreading, tanto a nivel del lenguaje (sintáctico) como por medio del sistema de runtime y objetos threads. Construír multithreading dentro del mismo lenguaje, brinda a los programadores una herramienta más fácil y poderosa para crear clases multithreaded seguras.

3.- Características eliminadas de C y C++ [25]

El primer paso fue la eliminación de redundancias de C y C++, ya que de alguna manera en C se encontraron muchas características superpuestas, proveyendo así distintas formas de hacer lo mismo.

- *No más typedefs, defines y preprocessor*

El código fuente escrito en Java es simple. No hay preprocesador, ni #define, ni capacidades relacionadas, ni typedefs, ni archivos header. En lugar de archivos header, los archivos fuente de Java proveen definiciones de otras clases y sus métodos.

Un problema importante de C es la cantidad de contexto que se debe conocer para entender el código de otro programador: antes de comenzar a analizar un programa, se deben leer todos los archivos header, los #define y los typedefs relacionados. En Java, se obtienen los efectos del typedef declarando clases. No se necesitan archivos header porque el compilador Java compila las definiciones de clases como códigos binarios que mantienen todo la información de los tipos durante el tiempo de linkeo.

Java es totalmente libre de contexto.

Los programadores pueden leer y entender el código, modificarlo y reusarlo mucho más rápida y fácilmente.

- *No más estructuras y uniones*

Java no tiene estructuras o uniones como tipos de datos complejos. No se necesitan porque se tienen clases, con las que se puede lograr el mismo efecto declarando variables de instancia como corresponde.

- *No más funciones*

Java no tiene funciones, ya que todo lo que puede hacerse con ellas también se puede hacer definiendo una clase y creando métodos para ella.

Esto no quiere decir que no sirvan los procedimientos y funciones, sino que, teniendo clases y métodos, se prefiere tener una sola forma de hacer las cosas.

- *No más herencia múltiple*

La herencia múltiple, junto con todos los problemas que genera, ha sido descartada de Java. Las características deseables de ella son provistas por las interfaces.

Una interface no es una definición de un objeto. Define un conjunto de métodos que estarán implementados por uno o más objetos. Las interfaces sólo declaran métodos y constantes, no se pueden definir variables en ellas.

- *No más instrucciones Goto*

Java no tiene la instrucción Goto. Su eliminación implicó una simplificación del lenguaje. Estudios hechos sobre aproximadamente 100.000 líneas de código C determinaron que el 90 % de las instrucciones goto utilizadas eran para lograr el corte de lazos anidados. Como se ve, los cortes multi-nivel y la instrucción continue eliminaron la mayor cantidad de sentencias goto.

- *No más operadores sobrecargados*

No hay forma de que los programadores puedan sobrecargar los operadores aritméticos standard. Los efectos de la sobrecarga de operadores pueden lograrse declarando una clase con variables de instancia y métodos apropiados para manejar esas variables.

- *No más coersiones aritméticas*

Java prohibe las coersiones automáticas de C y C++. Si se desea transformar un elemento de un tipo a otro, se provocará una pérdida de precisión, por lo tanto en Java debe hacerse explícitamente, usando un cast.

- *No más punteros*

Java no tiene punteros. Cualquier tarea que, en C, requiera de arreglos, de estructuras y de punteros, en Java se puede hacer de manera más fácil y confiable declarando objetos y arreglos de ellos. Se puede acceder a los arreglos a través de sus índices. El sistema de runtime de Java chequea todos los índices para asegurarse de que se encuentren dentro de los límites del arreglo.

Ya no es necesario el cuidado extremo para manejar punteros ni el miedo a corromper la memoria debido a punteros incorrectos, porque en Java no hay punteros.

4.- El problema de la superclase frágil [25]

En cualquier momento se pueden agregar nuevos métodos o variables de instancia a una clase; cualquiera y todas las clases que la referencien requerirán de una recompilación o se suspenderán. Las dependencias entre definiciones de clases y sus clientes proveen una fuente fructífera de errores de programación en C++, aún con la ayuda de utilitarios. La superclase frágil algunas veces está relacionada al problema constante de recompilación. En C++ se puede evitar este problema con ciertas dificultades, ésto implica no usar en forma directa cualquiera de las características orientadas a objetos del lenguaje.

El lenguaje Java resuelve el problema de la superclase frágil en varios niveles. El compilador Java no compila referencias bajo valores numéricos, además, pasa referencias de información simbólica a través del byte verificador de código y del intérprete. El intérprete Java ejecuta el nombre de la resolución final cuando se linkean las clases. Inmediatamente a que se resuelve el nombre, la referencia se reescribe como un offset numérico, habilitando al intérprete de Java a correr a velocidad completa.

Finalmente, la disposición de almacenamiento de objetos no está determinada por el compilador. La disposición de objetos en memoria difiere del tiempo de ejecución y

está determinada por el intérprete. La actualización de clases con nuevas instancias o métodos puede ser linkeada sin afectar el código existente.

Así, al no tener que realizar una recompilación constante, el lenguaje Java elimina el problema de la superclase frágil. Las librerías pueden agregar nuevos métodos y variables de instancia sin afectar a sus clientes.

5.- Seguridad [27]

5.1.- Introducción

El lenguaje Java permite a los browsers compatibles con Java traer fragmentos de código dinámicamente y luego ejecutarlos localmente. Sin embargo, los usuarios deben tener cuidado con el código que ejecutan porque podría venir de fuentes no confiables o podría haber pasado por redes inseguras.

Desde el momento en que el código compilado del lenguaje Java está diseñado para ser transportado en formato binario a través de las redes, la seguridad es sumamente importante.

Debido a que los bytecodes de Java corren en el host, hay ciertos requerimientos de seguridad. Los usuarios que bajan archivos de clase de Java de lugares remotos y, posiblemente, inseguros, deben saber que el código traído no puede destruír al intérprete de bytecodes de Java para realizar operaciones no permitidas.

5.2.- Verificación de los archivos de clase

Todos los archivos de clase traídos desde afuera están sujetos a un verificador. Este asegura que un archivo tenga el formato correcto. Los bytecodes son verificados usando un teorema simple, el cual establece un conjunto de requerimientos estructurales sobre los mismos.

El verificador de bytecodes también mejora la performance del intérprete. Se pueden eliminar los chequeos en tiempo de ejecución, que puedan realizarse en cada instrucción interpretada. Además, el intérprete puede asumir que estos chequeos ya se han realizado.

El verificador es independiente del compilador de Java. Aunque el intérprete certificará todo el código generado por el compilador, debería también certificar el que, posiblemente, no pueda generar. El verificador verificará cualquier conjunto de bytecodes que satisfaga el criterio estructural. El verificador es extremadamente conservativo. Rechazará certificar algunos archivos de clase que otros teoremas más sofisticados sí los aceptará.

5.3.- El formato de los archivos de clase

Cada archivo de clase Java es traído de la red como una entidad separada. El archivo de clase es simplemente un stream de bytes de 8 bits. Las cantidades de 16 bits y

32 bits se forman leeyendo dos o cuatro bytes de 8 bits respectivamente, y luego juntándolos en un formato más grande.

Un archivo de clase contiene:

- una constante mágica
- información de versiones mayor y mínima
- un *constant pool*
- información sobre la clase (nombre, superclase, etc.)
- información sobre cada uno de los campos y métodos de la clase
- información para el debugging

El *constant pool* es un arreglo de datos heterogéneos. Cada entrada puede tener uno de los siguientes elementos:

- un string de Unicodes
- un nombre de clase o interface
- una referencia a un campo o método
- un valor numérico
- un valor constante de tipo string

Ninguna otra parte del archivo de clase hace referencias específicas a strings, clases, campos o métodos.

Por cada campo y método de la clase, los bytes de ese archivo de clase indican el nombre del campo o método y su tipo. El tipo de un campo o método está indicado por un string: su signatura. Los campos pueden tener un atributo adicional que contiene su valor inicial. Los métodos también pueden tener un atributo adicional que indique el código para llamarlo.

Los métodos pueden tener muchos atributos con códigos. El atributo CODE indica el bytecode que se correrá en el intérprete. Podrían tener también atributos tales como SPARC-CODE o 386-CODE que son implementaciones de código de máquina del método.

5.4.- Los bytecodes y la máquina virtual

El atributo CODE provee información para la ejecución de métodos en lenguaje de máquina de la máquina virtual. La información para cada método incluye:

- el espacio máximo necesario para el método
- el número máximo de registros usados por el método
- el código actual para ejecutar el método. Los bytecodes son para la máquina virtual de Java
- una tabla de manejadores de excepciones. Cada entrada en la tabla tiene un puntero de inicio y uno de fin de los bytecodes, el tipo de excepción, y un desplazamiento para el manejador de la misma. La entrada indica que, si ocurre una excepción del tipo indicado dentro del código indicado por los

desplazamientos de inicio y fin, se encontrará un manejador de excepción en el desplazamiento dado para el manejador.

La máquina virtual de Java define 6 tipos primitivos:

- enteros de 32 bits ("integers")
- enteros de 64 bits ("longs" o "long integers")
- números en punto flotante de 32 bits ("single floats")
- números en punto flotante de 64 bits ("double floats")
- punteros a objetos y arreglos ("handles")
- punteros al código de la máquina virtual ("return address")

La máquina virtual también define varios tipos de arreglos, éstos incluyen arreglos de enteros, longs, single floats, double floats, handles, booleans, bytes (enteros de 8 bits), shorts (enteros de 16-bits) y caracteres Unicode.

Cada método de activación tiene una pila de evaluación separada y un conjunto de registros locales. La ubicación de cada registro y de cada pila debe poder guardar un entero, un single float, o una dirección de retorno. Los longs y los double float deben entrar en dos ubicaciones consecutivas de la pila o en dos registros consecutivos. Las instrucciones de la máquina virtual ("opcodes") direccionarán a longs y double floats en registros que usen un índice de registros de baja numeración.

El conjunto de instrucciones de la máquina virtual provee opcodes para operar sobre diferentes tipos de datos primitivos.

Las instrucciones de bytecodes pueden dividirse en varias categorías:

- apilar constantes en la pila
- acceder y modificar valores de un registro
- acceder a arreglos
- manipular pilas (por ejemplo pop, push, swap, etc)
- instrucciones aritméticas, lógicas y de conversión
- transferencias de control
- funciones de retorno
- manipular campos de objetos
- invocar métodos
- crear objetos
- casting de tipos

5.5.- El proceso de verificación

El verificador opera en 4 pasos:

Paso 1: El primer paso es el más simple. Ocurre cuando la clase es lo primero que lee el intérprete. Este paso asegura que el archivo de clase tiene el formato de un archivo de clase. Los primeros bytes deben contener el número mágico correcto. Todos los atributos reconocidos deben ser de la longitud apropiada. El archivo de clase no debe

haber sido truncado o tener bytes de más al final. El *constant pool* no debe contener información no reconocida.

Paso 2: Los errores detectados por este paso incluyen:

- Asegurar que las clases finales no tengan subclases, y que los métodos finales no se sobreescriban.
- Chequear que cada clase (excepto Object) tenga una superclase.
- Asegurar que el *constant pool* satisfaga ciertos requerimientos. Por ejemplo, que las referencias a clases en el *constant pool*, contengan un campo que apunte a una referencia a un string de Unicode del *constant pool*.
- Chequear que todas las referencias a campos y a métodos en el *constant pool* tengan nombres, clases y un tipo de signatura legales.

Paso 3: Se verifican los bytecodes de cada método. El verificador asegura que en cualquier punto dado del programa, sin importar qué camino se tomó para llegar a ese punto, se cumpla que:

- La pila esté siempre del mismo tamaño y contenga el mismo tipo de objetos.
- Ningún registro sea accedido a menos que se sepa que contiene un valor del tipo apropiado.
- Los métodos sean llamados con los argumentos apropiados.
- Los campos sean modificados con valores de los tipos apropiados.
- Todos los opcodes tengan el tipo de argumentos apropiados en la pila y en los registros.

Paso 4: Por razones de eficiencia, ciertos testeos que se pueden hacer en el Paso 3 son retrasados hasta que el código esté corriendo. El paso 3 del verificador evita cargar archivos de clase a menos que tenga que hacerlo.

Por ejemplo, si un método contiene una llamada a otro método que retorna un objeto de tipo foobarType y luego se asigna a otro objeto de ese mismo tipo, el verificador no se preocupa en verificar si el tipo foobarType existe. Sin embargo, si fuera asignado a un campo de tipo anotherType, se deben cargar las definiciones tanto de foobarType como de anotherType para asegurarse que foobarType sea una subclase de anotherType.

La primera vez que es ejecutada una instrucción que referencia a una clase, el verificador hace lo siguiente:

- Carga la definición de la clase si no ha sido aún cargada.
- Verifica que la clase que se está ejecutando actualmente tenga permiso de referenciar a la clase dada.

La primera vez que una instrucción llama a un método o accede o modifica un campo, el verificador hace lo siguiente:

- Asegura que exista el campo o método en la clase dada.
- Chequea que el método o campo tenga la signatura apropiada.

- Chequea que el método que se está ejecutando actualmente tenga acceso al método o campo dado.

Después de que se haya realizado la verificación, la instrucción es reemplazada en el stream del bytecode por una forma alternativa de dicha instrucción. Por ejemplo, el opcode new es reemplazado por new quick. Esta instrucción alternativa indica que ya ha ocurrido la verificación de esta instrucción, y no necesita realizarse de nuevo.

5.6.- El verificador de bytecodes

El paso 3 del verificador, el verificador de bytecodes, es el más complejo de la verificación de clases.

Primero, los bytes que forman las instrucciones virtuales son divididos en una secuencia de instrucciones, y el desplazamiento de inicio de cada una se guarda en una tabla de bits. Luego, el verificador recorre los bytes por segunda vez y hace un parse de las instrucciones. Durante este paso, cada una de ellas se convierte en una estructura. Los argumentos de cada una, si los hay, son chequeados para asegurarse que son razonables:

- Todas las instrucciones de control de flujo van al comienzo de una instrucción. No se permiten saltos al medio de una instrucción. Tampoco se permiten saltos hacia antes del comienzo o hasta después de la finalización de la misma.
- Todas las referencias a registros deben ser a uno legal. El código no puede acceder o modificar más registros que la cantidad que indica el método.
- Todas las referencias al *constant pool* deben ser una entrada del tipo apropiado.
- El código no termina en el medio de una instrucción.
- Para cada manejador de excepción, el punto de comienzo y fin debe apuntar al comienzo de una instrucción.

Para cada instrucción, el verificador guarda la pista de los contenidos de la pila y de los registros previos a la ejecución de esa instrucción. Para la pila, necesita saber su longitud y el tipo de cada uno de sus elementos. Para cada registro, necesita conocer el tipo de cada uno de los contenidos de ese registro o si contiene algún valor ilegal. El verificador de bytecodes no necesita distinguir entre los diferentes tipos de enteros (byte, short,char) cuando determina los tipos de valores de la pila.

Luego, se inicializa un analizador del flujo de datos. Para la primera instrucción, los registros de numeración más baja contienen los tipos indicados por la signatura de tipos de los métodos; la pila está vacía. Todos los otros registros contienen un valor ilegal. Para todas las otras instrucciones, se indica que la instrucción aún no ha sido visitada; todavía no hay información en su pila o registros.

Finalmente, se corre el analizador del flujo de datos. Para cada instrucción, hay un bit de cambio que indica si necesita ser buscada. Inicialmente, el bit de cambio está seteado sólo para la primera instrucción. El analizador del flujo de datos ejecuta el siguiente ciclo:

1.- Encontrar una instrucción de máquina virtual en la cual esté seteado el bit de cambio. Si ninguna instrucción lo conserva seteado, el método ha sido chequeado exitosamente y se apaga el bit de cambio.

2.- Emular el efecto de esta instrucción en la pila y los registros:

- Si la instrucción usa valores de la pila, asegura que haya suficientes elementos en la pila y que los elementos del tope de la misma sean del tipo apropiado. De otra forma, falla.
- Si la instrucción usa un registro, asegura que el registro especificado contenga un valor del tipo apropiado. De otra forma, falla.
- Si la instrucción apila valores en la pila, agrega los tipos especificados en el tope de la misma. Asegura que haya suficiente espacio en la pila para el nuevo elemento.
- Si la instrucción modifica un registro, indica que éste ahora contiene al nuevo tipo.

3.- Determinar las instrucciones de máquina virtual que pueden seguir a la actual. Las instrucciones sucesoras pueden ser una de las siguientes:

- la próxima instrucción, si la instrucción actual no era un goto, un return o un throw
- el destino de un salto condicional o incondicional
- todos los manejadores de excepciones para esta instrucción

4.- Mezclar el estado de la pila y de los registros al final de la instrucción actual en cada una de las instrucciones siguientes. En el caso del manejador de excepciones, se cambia la pila para que contenga un solo objeto del tipo de la excepción indicado por la información del manejador de excepciones.

- Si la instrucción sucesora se ha visitado por primera vez, indica que los valores de la pila y de los registros que se calcularon en el paso 2 y 3, son el estado de la pila y de los registros antes de ejectuar la instrucción sucesora; setear un bit de cambio para la instrucción sucesora.
- Si la instrucción ya ha sido visitada, mezclar los valores de la pila y de los registros calculados en los pasos 2 y 3 con los valores que ya están allí; setear el bit de cambio si hay alguna modificación.

5.- Volver al paso 1.

Para mezclar dos pilas, el número de elementos en cada una debe ser el mismo. Deben ser idénticas, excepto que aparezcan manejadores de distintos tipos en los lugares correspondientes de las dos pilas. En este caso, la pila mezclada contiene el ancestro común de los dos tipos de manejadores.

Para mezclar dos registros de estado, se deben comparar entre ellos. Si los dos tipos no son idénticos, entonces, a menos que los dos contengan manejadores, se debe indicar que el registro contiene un valor desconocido.

Si el analizador del flujo de datos corre sobre un método sin reportar fallos, entonces el método ha sido verificado exitosamente por el paso 3 del verificador de archivos de clase.

6.- Comparaciones entre el lenguaje Java y otros

Existen cientos de lenguajes de programación para que los desarrolladores escriban programas para solucionar problemas en distintas áreas.

Los lenguajes del nivel de Shell y Tcl, por ejemplo, son lenguajes de alto nivel totalmente interpretados. Algunos de ellos son apropiados para prototipos rápidos ya que fácilmente se pueden desarrollar ideas, agregarle nuevas aproximaciones y eliminarle aproximaciones que no andan, sin invertir demasiado tiempo en este proceso.

Los lenguajes scripting son también altamente portables. Su principal desventaja es la performance, ellos generalmente son mucho más lentos que las instrucciones de máquina naturales o los bytecodes.

En el medio existen lenguajes como Perl, que comparten muchas características con Java. La evolución de Perl lo ha llevado a adoptar características orientadas a objetos, de seguridad, etc. Algunas características comunes con Java pueden ser: comportamiento dinámico, robustez, arquitectura neutral, etc.

En el nivel más bajo están los lenguajes compilados como C y C++, en los cuales se pueden desarrollar programas de gran escala con alta perfomance, pero esto tiene su costo: el manejo de memoria es poco confiable y las capacidades de multithreading son difíciles de usar e implementar. Además está el problema de las superclases frágiles.

Java crea un término medio atractivo entre los lenguajes de alto nivel, portables, pero lentos como los de scripting y los lenguajes de bajo nivel, rápidos, pero no portables como los compilados. Java provee un buen nivel de performance para la mayoría de las aplicaciones con computaciones intensivas.

Java, Perl y Smalltalk son ambientes de programación comparables que ofrecen un conjunto rico de capacidades para los desarrolladores de aplicaciones de software.

7.- Un beneficio de Java: prototipación rápida y segura

Los lenguajes muy dinámicos como Lisp, Tcl y Smalltalk son usados para prototipación. Una de las razones de su éxito es que son muy robustos (no hay que preocuparse por la liberación o ruptura de la memoria).

De la misma forma, los programadores pueden estar seguros acerca del manejo de memoria cuando programan en Java. El sistema de recolección de residuos hace que el trabajo del programador sea mucho más fácil.

Otra razón por la que Lisp, Tcl y Smalltalk son buenos para la prototipación es que no requieren de la toma de decisiones temprana.

Java tiene justamente la propiedad opuesta: fuerza al programador a hacer elecciones explícitas. A partir de ellas se le brinda asistencia, ya que por ejemplo, si se escribe una invocación a un método y hay algo mal, el error se reporta en tiempo de compilación. No hay que preocuparse por errores en la invocación a métodos.

8.- Estructura de programas [28]

El código fuente de un programa Java consiste de una o más unidades de compilación. Cada una de ellas puede contener sólo lo siguiente:

- Una instrucción package.
- Instrucciones import.
- Declaraciones de clase.
- Declaraciones de interface.

Aunque cada unidad de compilación Java puede contener múltiples clases o interfaces, solamente puede ser pública una de ellas. En la implementación actual de Java, cada unidad de compilación es un archivo con el sufijo ".java".

8.1.- Clases

Las clases representan el modelo clásico de programación orientada a objetos. Soportan abstracción de datos e implementación asociada a ellos. En Java, cada clase nueva crea un tipo nuevo.

Para hacer una clase nueva, el programador debe basarse en una existente. Entonces, la clase nueva se dice derivada de la existente. También es llamada subclase de la otra que sería la superclase. La derivación de clases es transitiva.

La superclase inmediata de una clase y la interface que implementa, son indicadas en la declaración por las palabras claves: extends e implements como sigue:

```
class [nombre de la clase]
    extends [nombre de la superclase]
    implements [nombre de la interface 1, nombre de la interface 2, ...]
{ cuerpo de la clase }
```

Todas las clases son derivadas de la clase raíz: Object. Todas las clases, excepto Object tienen una superclase. Si una clase es declarada sin especificar su superclase, entonces se asume que Object es su superclase inmediata.

El lenguaje sólo soporta herencia simple, pero tiene algunas características de la herencia múltiple a través de las interfaces.

8.1.1.- Casting entre tipos clase

El lenguaje soporta casting entre tipos y, debido a que cada clase es un nuevo tipo, Java soporta casting entre clases. Si B es una subclase de A, entonces una instancia

de B puede usarse como instancia de A. En este caso el cast es implícito, aunque el cast explícito es legal (esto se llama widening). Si una instancia de A necesita ser usada como si fuera instancia de B, el programador puede escribir un cast (esto se llama narrowing). El cast de una clase a sus subclases es siempre chequeado en tiempo de ejecución para asegurarse que el objeto actualmente, sea una instancia de la subclase. El casting entre clases hermanas da un error de compilación. La sintaxis del cast de una clase es:

(nombre de la clase) referencia

donde (nombre de la clase) es el objeto hacia el cual se hace el cast y referencia es el objeto que está siendo convertido.

El casting afecta sólo a la referencia al objeto y no al objeto mismo. Sin embargo, el acceso a las variables de instancia está afectado por el tipo de la referencia al objeto. El casting entre un objeto de un tipo a otro tipo puede implicar que se referencien variables de instancia diferentes aunque se esté usando el mismo nombre de variable. Por ejemplo:

```
class ClaseA
   String name = "claseA"

class ClaseB
    extends ClaseA
    String name = "claseB"

class ClaseC
   void test() {
      ClaseB b = new  ClaseB()
      println (b.name)  // imprime claseB

      ClaseA a;
      a = (ClaseA) b
      println (a.name)  // imprime claseA

      b = (ClaseA) b
      println (b.name) // imprime claseA aunque se llame igual que la b de la derecha del
                          igual que imprimió claseB
```

8.1.2.- Métodos

Los métodos son operaciones que se pueden ejecutar sobre un objeto o una clase. Pueden declararse tanto en clases como en interfaces, pero sólo pueden ser implementados en clases.

La declaración de un método en una clase tiene la siguiente forma:

```
[comentarios] [modificadores] tipoDeRetorno nombreDelMétodo ( listaDeParámetros) {
    [cuerpo del método]
}
```

Los métodos:

- Tienen un tipo de retorno a menos que se trate de métodos constructores. Si un método no constructor no tiene un valor de retorno, entonces se debe poner como tipo de retorno la palabra void.

- Tienen una lista de parámetros que consiste de pares tipo-nombre del parámetro, separados por comas. Si el método no tiene parámetros la lista estará vacía.

Las variables declaradas en los métodos no pueden ocultar a otras variables o parámetros del mismo método, es decir, no se puede usar nombres de variables locales repetidos ni iguales a los nombres de los parámetros.

El lenguaje permite polimorfismo, es decir, declarar un método con un nombre que ya fue usado en esa clase o en su superclase, mediante la sobreescritura (overriding) o sobrecarga (overloading) del método. Overriding significa que a un método heredado, se lo provea de una implementación diferente. Overloading significa declarar un método con el mismo nombre que otro pero con diferente lista de parámetros.

El tipo de retorno no es utilizado para distinguir métodos. Dentro del alcance de una clase, los métodos que tienen el mismo nombre y lista de parámetros, deben retornar el mismo tipo, en caso contrario da un error de compilación.

8.1.2.1.- Variables de instancia

Todas las variables de una clase que están fuera del alcance de un método y que no están marcadas como estáticas, son variables de instancia. Las variables declaradas dentro del alcance de un método se llaman locales.

Las variables de instancia pueden tener modificadores. Además, pueden ser de cualquier tipo y tener inicializadores. Si no tienen un inicializador, entonces son inicializadas en 0; las variables booleanas son inicializadas en false y los objetos en null.

8.1.2.2.- Variables this y super

Dentro del alcance de un método no estático, el nombre representa al objeto corriente. Cada vez que un método referencia a sus propias variables de instancia o a sus propios métodos, se considera que un "this" está en frente de esa referencia.

La variable "super" es similar a "this", pero, "this" contiene una referencia al objeto corriente y su tipo es la clase que contiene el método que se está ejecutando, mientras que la variable "super" contiene una referencia del tipo de la superclase.

8.1.3.- Constructores

Los constructores son métodos especiales que sirven para inicialización. Tienen el mismo nombre que la clase pero no tienen tipo de retorno. Son llamados automáticamente en la creación de un objeto (no pueden ser llamados explícitamente por un objeto, y si se los quiere llamar fuera del paquete, el constructor debe declararse como público).

Los constructores pueden ser sobrecargados variando el número y tipo de los parámetros, al igual que cualquier otro método.

Las variables de instancia de las superclases se inicializan llamando a un constructor para la superclase inmediata o a uno de la clase corriente. Si no se especifica ninguno, se invoca al constructor de la superclase que no tenga parámetros. Si un constructor llama a otro en la clase corriente o en la superclase inmediata, ese llamado debe ser lo primero que se realice. Las variables de instancia no pueden referenciarse antes de la llamada al constructor. Si una clase no declara constructores, el compilador le genera uno automáticamente.

8.1.4.- Creación de objetos: el operador new

Una clase es un patrón usado para definir el estado y comportamiento de un objeto. Un objeto es una instancia de una clase. Todas las instancias de las clases están alocadas en un depósito del recolector de residuos. No se aloca espacio al declarar una referencia a un objeto. El programador debe alocarlo explícitamente, pero no es necesaria la desalocación explícita, ya que el recolector de residuos reclama automáticamente la memoria cuando no se usa más.

Para alocar espacio para un objeto, se usa el operador new. Este inicializa las variables de instancia y luego llama al constructor de la instancia. La siguiente sintaxis aloca e inicializa una nueva instancia de una clase llamada ClaseA:

```
a = new ClaseA()
```

Esta otra sintaxis provee argumentos al constructor:

```
b = new ClaseA(3,2)
```

Una tercera forma de alocación permite que el nombre de la clase sea una expresión String, éste es evaluado en tiempo de ejecución y el new retorna un objeto de tipo Object, el cual debe ser convertido por un cast al tipo deseado:

```
b = new ("Clase"+"A")
```

En este caso se está invocando a un constructor sin argumentos.

8.1.5.- Especificadores de acceso

Los especificadores de acceso son modificadores que permiten a los programadores acceder a métodos y variables. Las palabras claves usadas son: public, private y protected.

Los métodos marcados como públicos, pueden ser accedidos desde cualquier lugar y por cualquier persona. Los marcados como private pueden ser accedidos solamente dentro de la clase en la que fueron declarados, por lo tanto no son visibles fuera de la clase, por lo que no pueden sobreescribirse. Tampoco se puede sobreescribir un método no private y darle acceso privado. Los especificadores de acceso protected hacen que las variables o métodos definidos en una clase sean accesibles por la subclase pero no por otras clases.

El acceso público puede aplicarse a clases, métodos y variables, y en este caso, éstas pueden ser accedidas desde cualquier lugar por otras clases o métodos. El acceso de un método público no puede cambiarse sobreescribiéndolo.

Las clases, métodos y variables a las que no se les especificó una forma de acceso, pueden ser accedidas sólo dentro del paquete donde fueron declaradas.

8.2.- Interfaces

Una interface especifica una colección de métodos sin sus cuerpos de implementación. Las interfaces proveen encapsulamiento de protocolos de métodos sin restringir la implementación a un árbol de herencia. Cuando una clase implementa una interface, generalmente debe implementar el cuerpo de todos los métodos descriptos en ella.

Las interfaces resuelven algunos de los problemas que tiene la herencia múltiple sin demasiada sobrecarga en tiempo de ejecución. Sin embargo, debido a que las interfaces involucran binding dinámico de métodos, al usarlas se produce una pequeña falla en la performance.

Las clases e interfaces pueden ser privadas o públicas. El alcance de las interfaces públicas o privadas es el mismo que el de las clases públicas o privadas respectivamente. Los métodos en una interface son siempre públicos. Las variables son públicas, estáticas y finales.

8.2.1.- Interfaces como tipos

La sintaxis de declaración: [nombre_interface] [nombre_variable] declara una variable o parámetro como instancia de alguna clase que implementa nombre_interface. Las interfaces se comportan exactamente como las clases cuando son usadas como un tipo. Esto permite al programador especificar que un objeto debe implementar una interface dada, sin tener que conocer el tipo o herencia exacto de ese objeto. El uso de interfaces hace innecesario forzar, a clases relacionadas, a compartir una superclase abstracta común o agregar métodos a Object.

8.2.2.- Métodos en interfaces

Los métodos en las interfaces se declaran como sigue:

nombre_tipo_retorno nombre_método (lista de parámetros).

Las declaraciones no contienen modificadores. Todos los métodos especificados en una interface son públicos y abstractos, y no debe aplicarse ningún modificador.

8.2.3.- Variables en interfaces

Las variables declaradas en interfaces son finales, públicas y estáticas, no se pueden aplicar modificadores. Dichas variables deben inicializarse.

8.2.4.- Combinación de interfaces

Las interfaces pueden incorporar una o más interfaces usando la palabra clave *extends* como sigue:

```
interface Hacer extends Almacenar, Pintar {
        void hacerAlgo () ;
}
```

8.3.- Package

Los package son grupos de clases e interfaces. Son una herramienta para manejar un gran espacio y para evitar conflictos. Cada nombre de clase o interface está contenido en algún package. Por convención, los nombres de los packages consisten de palabras separadas por puntos, con el primer nombre representando a la organización que desarrolló el package.

8.3.1.- Especificación del package de la unidad de compilación

El package en el que está una unidad de compilación está especificado por una sentencia package. Cuando esta instrucción está presente, debe ser la primer línea sin espacios en blanco ni comentarios. Tiene el siguiente formato:

```
package nombre_del_package
```

Cuando una unidad de compilación no tiene sentencia package, es ubicada en un package por default, el cual no tiene nombre.

8.3.2.- Uso de clases e interfaces desde otros packages

El lenguaje provee un mecanismo para hacer que las definiciones e implementaciones de clases o interfaces estén disponibles a través del package. La palabra clave *import* se usa para marcar las clases que serán importadas en el package actual. Una unidad de compilación importa automáticamente cada clase e interface de su propio package.

El código de un package puede especificar clases e interfaces de otro en una de estas dos formas:

- Anteponiendo el nombre del package a cada referencia a un nombre de clase o interface. Por ejemplo: acme.project.FooBar obj=new acme.project.FooBar()

- Importando la clase, interface o package que lo contiene, usando la instrucción import. Al importar una clase o interface el nombre de ella estará disponible en el espacio actual. Al importar un package los nombres de todas las clases públicas o interfaces estarán disponibles. Por ejemplo: import acme.project.*; significa que se importan todas las clases públicas de acme.project. El siguiente constructor importa una clase simple, ListaEmpleados, desde el package acme.project:
  ```
  import acme.project.ListaEmpleados;
  ListaEmpleados obj=new ListaEmpleados();
  ```

Es ilegal especificar un nombre de clase ambiguo, lo que siempre genera un error en tiempo de compilación. Los nombres de clase pueden ser desambiguados a través del uso de un nombre de clase totalmente calificado, por ejemplo, uno que incluya el nombre del package de la clase.

Apéndice B

Invocación de Métodos Remotos (RMI) [30]

1.- Arquitectura de Sistemas

El sistema RMI consta de tres capas: la capa stub/skeleton, la capa de referencias remotas y la capa de transporte. El límite entre cada una está definido por una interface y un protocolo específicos; además, cada capa es independiente de la siguiente y puede ser reemplazada por una implementación alternativa sin afectar a las otras capas del sistema. Por ejemplo, la implementación actual de la capa de transporte está basada en TCP (usando sockets Java), pero podría ser reeemplazada por un transporte basado en UDP.

Para acompañar la transmisión transparente de objetos de un espacio de direcciones a otro, se usa la técnica de serialización de objetos (diseñada específicamente para Java).

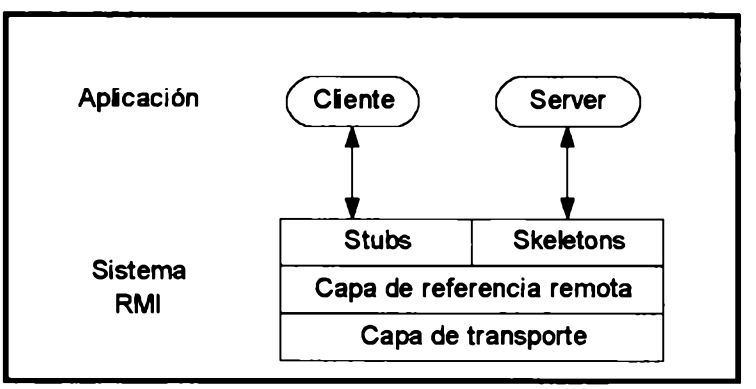
Otra técnica, llamada carga dinámica de stubs, es usada para soportar los stubs del lado del cliente, los cuales implementan el mismo conjunto de interfaces remotas que el objeto remoto mismo. Esta técnica, usada cuando un stub del tipo exacto ya no está disponible en el cliente, permite a éste usar los operadores predefinidos de Java para casting y chequeos de tipos.

1.1.- Visión arquitectural

El sistema RMI consta de tres capas:

- *La capa stub/skeleton:* los stubs están del lado del cliente y los skeleton del lado del server.
- *La capa de referencias remotas:* es el comportamiento de las referencias remotas (por ejemplo, la invocación a un objeto simple o a uno replicado).
- *La capa de transporte:* seteo y manejo de conecciones y seguimiento de objetos remotos.

La capa de aplicación se sitúa en el tope del sistema RMI. La relación entre las capas se puede ver en la siguiente figura:



Una invocación de un método remoto desde un cliente a un objeto server remoto viaja hacia abajo por las capas del sistema RMI hasta la capa de transporte del lado del cliente, y luego hacia arriba a través de la capa de transporte del lado del server.

Un cliente que invoca un método sobre un objeto server remoto hace uso de un stub o representante del objeto remoto, como si fuera un conducto hacia él. Una referencia a un objeto remoto que mantiene un cliente, es un referencia a un stub local. Este stub es una implementación de las interfaces remotas del objeto remoto y envía pedidos de invocaciones a ese objeto server vía la capa de referencias remotas. Los stub y los skeleton se generan usando el compilador rmic.

La capa de referencias remotas es la responsable de llevar a cabo la semántica de la invocación. Por ejemplo, es la responsable de determinar si el server es un objeto simple o un objeto replicado que requiere comunicaciones con múltiples ubicaciones. La implementación de cada objeto remoto elije su propia semántica de referencias remotas (si el server es un objeto simple o es un objeto replicado que requiere comunicaciones con sus réplicas).

La semántica de referencias para el server también está manejada por la capa de referencias remotas. Esta, por ejemplo, abstrae las diferentes formas de referenciar a objetos que están implementados en:

- Servers que están siempre corriendo en alguna máquina.
- Servers que corren únicamente cuando se invoca a alguno de sus métodos (activación).

Las capas de más arriba de las de referencias remotas, no notan estas diferencias.

La capa de transporte es la responsable del seteo y manejo de conecciones, y del seguimiento de y envío hacia objetos remotos (el destino de las llamadas remotas), que residen en los espacios de direcciones del transporte.

Para hacer envíos hacia objetos remotos, la capa de transporte manda las llamadas remotas hacia la capa de referencias remotas. Esta maneja cualquier comportamiento del lado del server que necesite efectuarse antes de que el pedido pase al skeleton del lado del server. El skeleton de un objeto remoto, llama a la implementación del objeto remoto la cual realiza la llamada al método actual.

El valor de retorno de una llamada es enviado a través del skeleton, la capa de referencias remota y la de transporte del lado del server, y luego sube a través de la capa de transporte, la de referencias remotas y el stub del lado del cliente.

1.2.- Capa stub/skeleton

La capa stub/skeleton es la interface entre la capa de aplicación y el resto del sistema RMI. Esta capa no tiene nada que ver con cosas especificas de ningún transporte, pero transmite datos a la capa de referencias remotas vía la abstracción de streams ordenados. Los streams ordenados emplean un mecanismo llamado serialización de objetos el cuál habilita a los objetos Java a ser transmitidos entre espacios de

direcciones. Los objetos transmitidos usando el sistema de serialización de objetos son pasados por copia al espacio de direcciones remoto, a menos que sean objetos remotos, en cuyo caso, son pasados por referencia.

Un stub de un objeto remoto es el representante del lado del cliente de ese objeto remoto. Tal stub implementa todas las interfaces que soporta la implementación del objeto remoto. Un stub del lado del cliente es el responsable de:

- Iniciar una llamada al objeto remoto (llamando a la capa de referencias remota).
- Guardar los argumentos en un stream ordenado (obtenidos de la capa de referencias remota).
- Informar a la capa de referencias remotas que se puede invocar la llamada.
- Recuperar el valor de retorno o excepciones del stream ordenado.
- Informar a la capa de referencias remotas que se ha completado la llamada.

El skeleton de un objeto remoto es una entidad del lado del server que contiene un método que envía llamadas a la implementación del objeto remoto actual. El skeleton es el responsable de:

- Recuperar argumentos del stream ordenado.
- Hacer la llamada a la implementación del objeto remoto actual.
- Guardar el valor de retorno de la llamada o la excepción (si es que ocurrió) dentro del stream ordenado.

Las clases stub y skeleton apropiadas se determinan en tiempo de ejecución y se cargan dinámicamente según se necesitan.

1.3.- Capa de referencias remotas

La capa de referencias remotas tiene que ver con la interface de transporte de menor nivel. Esta capa es también responsable de realizar un protocolo de referencias remotas específico el cual es independiente de los stubs del cliente y de los skeletons del server.

Cada implementación de objetos remotos elije sus propias subclases de referencias remotas que operan a su favor. En esta capa se pueden realizar varios protocolos de invocación, por ejemplo:

- Invocación unicast punto a punto.
- Invocación a grupos de objetos replicados.
- Soporte de una estrategia de replicación específica.
- Soporte de una referencia persistente a un objeto remoto (habilitando la activación del objeto remoto).
- Estrategias de reconección (si el objeto remoto está inaccesible).

La capa de referencias remota tiene dos componentes que cooperan: los componentes del lado del cliente y los del lado del server. Los del lado del cliente

contienen información específica del server remoto (o servers, si la referencia remota es a un objeto replicado) y se comunican vía la capa de transporte con los componentes del lado del server. Durante cada invocación a un método, los componentes del lado del cliente y del server realizan la semántica de referencias remotas específicas. Por ejemplo, si un objeto remoto es parte de un objeto replicado, el componente del lado del cliente puede enviar la invocación a cada réplica, en lugar de mandarla a un solo objeto remoto.

De la misma forma, el componente del lado del server implementa las semánticas de referencias remotas específicas antes de enviar una invocación de métodos remotos al skeleton. Este componente, por ejemplo, podría asegurar el envío de multicasts atómicos mediante la comunicación con otros servers en un grupo de réplica.

La capa de referencias remotas transmite datos a la capa de transporte a través de la abstracción de una conección orientada a streams. La capa de transporte tiene cuidado de los detalles de implementación de las conecciones. Aunque las conecciones presentan una interface basada en streams se puede implementar un transporte de baja conección por debajo de la abstracción.

1.4.- Capa de transporte

En general, la capa de transporte del sistema RMI es la responsable de:

- Setear conecciones con los espacios de direcciones remotos.
- Manejar conecciones.
- Monitorear la vida de las conecciones.
- Escuchar las llamadas entrantes.
- Mantener la tabla de objetos remotos, que reside en el espacio de direcciones.
- Setear una conección para una llamada entrante.
- Localizar al enviador de la llamada remota y pasarle la conección.

La representación concreta de una referencia de un objeto remoto consiste de un punto final y de un identificador de objetos. Esta representación se llama referencia viva. Dada una referencia viva para un objeto remoto, la capa de transporte puede usar el punto final para setear una conección con el espacio de direcciones en el cual reside dicho objeto remoto. En el lado del server, la capa de transporte usa el identificador del objeto para buscar el destino de la llamada remota.

La capa de transporte para el sistema RMI consiste de cuatro abstracciones básicas:

- Un punto final es una abstracción usada para denotar un espacio de direcciones o una máquina virtual Java. En la implementación, un punto final puede mapearse con su transporte. Esto es, dado un punto final, se puede obtener una instancia de un transporte específico.
- Un canal es la abstracción para un conducto entre dos espacios de direcciones. Como tal, es el responsable de manejar conecciones entre el espacio de direcciones local y el remoto.

- Una conección es la abstracción de las transferencias de datos (realizar entrada/salida)
- La abstracción transporte maneja canales. Cada canal es una conección virtual entre dos espacios de direcciones. Dentro de un transporte, existe solamente un canal por cada par de espacios de direcciones, el espacio de direcciones local y el remoto. Dado un punto final de un espacio de direcciones remoto, el transporte setea un canal hacia ese espacio. La abstracción transporte también es la responsable de aceptar llamadas sobre conecciones entrantes del espacio de direcciones, setear un objeto conección para la llamada y enviarlo a las capas de más arriba en el sistema.

Un transporte define cual es la representación concreta de un punto final, por lo cual pueden existir muchas implementaciones de transporte. El diseño y la implementación también soportan múltiples transportes por espacios de direcciones, así TCP y UDP pueden ser soportados en la misma máquina virtual.

1.5.- Uso de threads en RMI

Un método enviado por el runtime de RMI a una implementación de objeto remoto (un server) puede o no ejecutarse en un thread separado. Algunas llamadas originadas desde la misma máquina virtual del cliente se ejecutarán en el mismo thread, mientras que otras se ejecutarán en threads diferentes. Las llamadas originadas en diferentes máquinas virtuales de clientes se ejecutarán en threads diferentes. Más aún, el runtime de RMI no garantiza nada con respecto al mapeo de invocaciones de objetos remotos con threads.

1.6.- Recolección de residuos de objetos remotos

En un sistema distribuido, de la misma manera que en un sistema local, es deseable borrar automáticamente aquellos objetos remotos que ya no estén referenciados por ningún cliente. Esto libera al programador de la necesidad de seguir la pista de los objetos remotos de los clientes y así poder terminar en forma apropiada. RMI usa un algoritmo de recolección de residuos con un contador de las referencias.

Para acompañar a la recolección de residuos, el runtime de RMI mantiene la pista de todas las referencias vivas de cada máquina virtual Java. Cuando una referencia viva entra a una máquina virtual Java, se incrementa su contador de referencias. La primera referencia a un objeto envía un mensaje "referenced" al server del objeto. Debido a que las referencias vivas pueden dereferenciarse en la máquina virtual local, su finalización decrementa el contador. Cuando se descarta la última referencia, se envía al server un mensaje "unreferenced". Existen muchas sutilezas en el protocolo, la mayoría relacionadas con mantener el orden de los mensajes referenciados y dereferenciados para asegurar que el objeto no sea recolectado prematuramente.

Cuando un objeto remoto no está referenciado por ningún cliente, el runtime de RMI se refiere a él usando una referencia débil. Esta permite al recolector de residuos de la máquina virtual Java descartar al objeto si no tiene ninguna otra referencia local. El algoritmo de recolección de residuos distribuido interactúa con el recolector de residuos de la máquina virtual Java local en las formas usuales, manteniendo referencias débiles o

normales a los objetos. De la misma manera que en el ciclo de vida del objeto normal, se puede llamar a un finalize después de que el recolector de residuos determine que no existen más referencias al objeto.

Cuando existe una referencia local a un objeto remoto, éste no puede ser recolectado; tampoco puede ser pasado en llamadas remotas o retornado a los clientes. Al pasar un objeto remoto se agrega, al conjunto de referencias, el identificador de la máquina virtual a la cual fue pasado. Un objeto remoto que necesita una notificación "unreferenced" debe implementar la interface java.rmi.server.Unreferenced. Cuando estas referencias no existan más, se invocará al método "unreferenced", es decir, éste es llamado cuando el conjunto de referencias está vacío, por lo tanto puede ser llamado más de una vez. Los objetos remotos son recolectados únicamente cuando no existen más referencias a ellos, ya sean locales o remotas.

Notemos que si existe una partición de red entre un cliente y un objeto server remoto, es posible que ocurra una recolección prematura del objeto remoto (ya que la capa de transporte puede pensar que el cliente tuvo un crash). Debido a la posibilidad de recolecciones prematuras, las referencias remotas no pueden garantizar integridad referencial; en otras palabras, siempre es posible que una referencia remota pueda de hecho no referenciar a ningún objeto existente. Un intento de usar tal referencia generará una RemoteException que debe ser manejada por la aplicación.

1.7.- Carga dinámica de clases

En los sistemas RPC, el código de los stubs del lado del cliente debe ser generado y linkeado en el cliente antes de que se realice la llamada al procedimiento remoto. Este código puede ser linkeado estáticamente en el cliente o en tiempo de ejecución mediante un linkeo dinámico, usando librerías que estén disponibles localmente o a través del sistema de archivos de la red. En ambos casos, el código específico para manejar un RPC debe estar disponible en la máquina del cliente en forma compilada.

RMI generaliza esta técnica usando un mecanismo llamado carga dinámica de clases para cargar en tiempo de ejecución (en formato de bytecode de la arquitectura neutral de Java) las clases que se requieren a fin de manejar la invocación de métodos sobre un objeto remoto. Estas clases son:

- Las clases de los objetos remotos y sus interfaces.
- Las clases stub y skeleton que sirven como representantes de los objetos remotos.
- Otras clases usadas directamente en las aplicaciones basadas en RMI, tales como parámetros o valores de retorno de las invocaciones de métodos remotos.

1.7.1.- ¿Cómo se elije un cargador de clases?

En Java, el cargador de clases que inicialmente carga una clase Java es usado subsecuentemente para cargar todas las interfaces y clases que se usen directamente en la clase:

- El AppletClassLoader es usado para traer un applet Java a través de la red desde una ubicación especificada por el atributo codebase de la página WWW que contiene el rótulo <applet>. Todas las clases usadas directamente en el applet son cargadas subsecuentemente por el AppletClassLoader.

- Para cargar aplicaciones Java (tanto un cliente como un server) desde el CLASSPATH local, se usa el cargador de clases por defecto. Todas las clases usadas directamente en la aplicación son cargadas subsecuentemente por el cargador de clases por defecto desde el CLASSPATH local.

- El RMIClassLoader es usado para cargar aquellas clases que no son usadas directamente por la aplicación cliente o server: los stubs de los objetos remotos, las interfaces remotas y las clases extendidas de los argumentos y valores de retorno de llamadas RMI. El RMIClassLoader busca estas clases en las siguientes ubicaciones, en este orden:
  1. El CLASSPATH local. Las clases son siempre cargadas localmente si es que existen localmente.
  2. Para las referencias a objetos remotos pasados como parámetros o valores de retorno, en el URL codificado en el stream ordenado que contiene al objeto serializado.
  3. En el URL especificado por la propiedad java.rmi.server.codebase local.

Para el caso 2, el URL del objeto remoto es pasado en el stream sólo si el RMIClassLoader de la máquina virtual donde reside la implementación del objeto remoto cargó la clase desde la ubicación especificada por la propiedad java.rmi.server.codebase (o el AppletClassLoader cargó la clase desde el codebase del applet). Si la clase fue cargada desde el CLASSPATH, el URL no es enviado, ni aún habiéndose seteado la propiedad java.rmi.server.codebase.

La aplicación puede configurarse con la propiedad java.rmi.server.useCodebaseOnly, la cual deshabilita la carga de clases desde el host de la red y fuerza a que éstas sean cargadas únicamente a partir del codebase definido localmente. Si la clase requerida no se puede cargar, la invocación del método fallará con una excepción.

1.7.2.- Arranque del cliente

Para que el runtime de RMI sea capaz de traer todas las clases e interfaces que necesita una aplicación cliente, se requiere de un programa cliente de arranque, el cual fuerce al uso del RMIClassLoader en lugar del cargador de clases por defecto. El programa de arranque necesita:

- Crear una instancia del RMISecurityManager o de un manejador de seguridad definido por el usuario.

- Usar el RMIClassLoader para cargar el archivo de clase para el cliente. El nombre de la clase no puede mencionarse explícitamente en el código pero debe ser un string o un argumento de la línea de comandos. De otra forma, el

cargador de clases por defecto tratará de cargar el archivo de clase del cliente del CLASSPATH local.

- Usar el método newInstance para crear una instancia del cliente y hacer un cast a Runnable (el cliente debe implementar la interface java.lang.Runnable).

- Comenzar el cliente llamando al método run (heredado de Runnable).

Una vez que ha comenzado el cliente y tiene el control, todas las clases que necesita serán cargadas por el RMIClassLoader desde java.rmi.server.codebase. Esta técnica de arranque es la misma que usa Java para forzar al AppletClassLoader a traer las mismas clases usadas en un applet.

Sin esta técnica de arranque, todo el código que necesita el cliente debe estar disponible a través de su CLASSPATH local, y el único código Java que puede ser cargado por el RMIClassLoader desde la red son los archivos de clase que no se usan directamente en el programa cliente: las interfaces remotas, los stubs y las clases extendidas de los argumentos y valores de retorno de las invocaciones a métodos remotos.

1.8.- Seguridad

En Java, cuando un cargador de clases las carga desde el CLASSPATH local, se considera que son clases confiables y no estan restringidas por el manejador de seguridad. Sin embargo, cuando el RMIClassLoader intenta traer clases desde la red, debe haber un manejador se seguridad o se tirará una excepción.

El manejador de seguridad debe ser iniciado como la primer acción del programa Java de manera que pueda regular las acciones subsecuentes. Este asegura que las clases cargadas concuerden con las garantías de seguridad del standard de Java, por ejemplo que esas clases se cargen desde fuentes confiables (por ejemplo el host del applet) y no intenten acceder a funciones sencibles.

Las applets están siempre sujetas a restricciones impuestas por la clase AppletSecurity. Este manejador de seguridad asegura que las clases sean cargadas únicamente desde el hosts del applet o desde los host designados por el codebase. Esto requiere que los desarrolladores del applet instalen las clases apropiadas en el host de dicho applet.

Las aplicaciones deben definir su propio manejador de seguridad o usar el RMISecurityManager. Si no hay un manejador de seguridad, una aplicación no puede cargar clases desde la red.

Un programa cliente o server generalmente está implementado por clases cargadas desde el sistema local y por lo tanto no está sujeto a las restricciones del manejador de seguridad. Sin embargo, si el programa cliente es traído desde la red usando la técnica descripta en 10.9.7.2, entonces el programa cliente está sujeto a las restricciones del manejador de seguridad.

Una vez que una clase fue cargada por el RMIClassLoader, cualquier otra clase usada directamente por ella también será cargada y estará sujeta a las restricciones de ese manejador de seguridad.

Aún si existe un manejador de seguridad, setear la propiedad java.rmi.server.useCodebaseOnly a true previene de traer una clase del URL embebido en el stream con un objeto serializado (las clases aún pueden ser cargadas a partir del java.rmi.server.codebase definido localmente). La propiedad java.rmi.server.useCodebaseOnly puede especificarse tanto en el cliente como en el server, pero no es aplicable en applets.

Si una aplicación define su propio manejador de seguridad el cual no permite la creación de un cargador de clases, las clases serán cargadas usando el mecanismo por defecto Class.forName. Entonces, un server puede definir sus propias políticas vía el manejador de seguridad y el cargador de clases, y el sistema RMI operará con ellas.

La clase abstracta java.lang.SecurityManager, de la cual extienden todos los manejadores de seguridad, no regula la consumición de recursos. Así el RMISecurityManager actual no dispone de mecanismos para prevenir la carga de clases desde recursos abusivos.

1.9.- Configuración de escenarios

El sistema RMI soporta diferentes escenarios. Los servers pueden configurarse en forma abierta o cerrada. Las applets pueden usar RMI para invocar métodos sobre objetos soportados en servers. Si un applet crea y pasa un objeto remoto al server, éste puede usar RMI para devolver la llamada al objeto remoto. Las aplicaciones Java pueden usar RMI tanto en la forma cliente/servidor, como en la forma peer to peer.

1.9.1.- Servers

El escenario típico de un sistema cerrado configura al server de manera tal que no cargue clases. Los servicios que provee están definidos por las interfaces remotas, que son todas locales a la máquina del server. El server no tiene manejador de seguridad y no cargará clases, aún si los clientes envían el URL. Si los clientes envían objetos remotos para los cuales el server no tiene las clases stub, estas invocaciones a métodos fallarán cuando se recupere el pedido, y el cliente recibirá una excepción.

Un sistema de server más abierto definirá su java.rmi.server.codebase para que las clases de los objetos remotos que exporta puedan ser cargadas por los clientes, y además, para que el server pueda cargar clases para los objetos remotos provistos por los clientes, cuando las necesite. El server tendrá un manejador de seguridad y un cargador de clases RMI que lo protejerán. Un server más cuidadoso podría usar la propiedad java.rmi.server.useCodebaseOnly para deshabilitar la carga de clases desde las URLs provistas por los clientes.

1.9.2.- Applets

Generalmente, las clases necesitan ser provistas por un server HTTP o por uno FTP según lo referenciado en la URL embebida en la página HTML que contiene al applet. Los servicios basados en RMI que usa el applet deben estar en el server desde el cual fue traído, debido a que un applet únicamente puede hacer conecciones de red con el host desde el cual fue cargado.

Por ejemplo, un escenario normal para un applet usa un solo host para el server HTTP que provee la página HTML, el código del applet, los servicios RMI, y el Registry de arranque. En este escenario, todos los stubs, skeletons y clases de soporte son cargados desde el server HTTP. Todos los objetos remotos provistos por el servicio RMI y pasados al applet (los cuales pueden pasar de vuelta al server) serán para clases que el servicio RMI ya conozca. En este caso, el servicio RMI es muy seguro porque no carga clases de la red y por lo tanto no necesita un manejador de seguridad.

1.9.3.- Aplicaciones

Las aplicaciones escritas en Java, a diferencia de los applets, pueden conectarse a cualquier host, de manera que tienen más opciones disponibles para configurar las fuentes de las clases y donde van a correr los servicios basados en RMI. Generalmente, será usado un solo server HTTP para proveer las clases remotas, mientras que las aplicaciones basadas en RMI estarán distribuidas por la red sobre servers o corriendo sobre desktops de usuarios.

Si una aplicación es cargada localmente, entonces, las clases que se usan directamente en ese programa también deben estar disponibles localmente. En este escenario, las únicas clases que pueden ser traídas desde una fuente de red, son las de las interfaces remotas, las de los stubs y las extendidas de los argumentos y valores de retorno de invocaciones a métodos remotos.

Si una aplicación no es cargada desde el directorio local pero sí lo es desde una fuente de red usando el mecanismo de arranque descripto en 10.9.7.2, entonces todas las clases usadas por la aplicación pueden ser traídas desde la misma fuente de red.

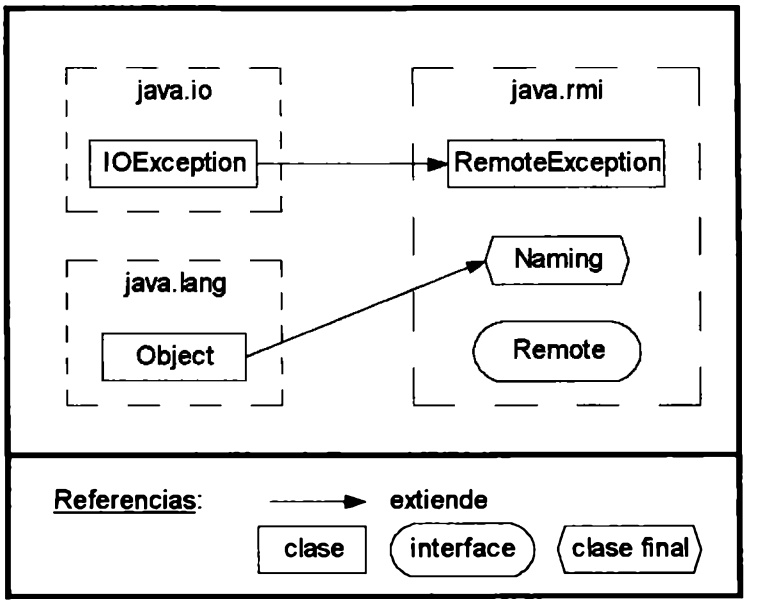
Para habilitar la carga desde una fuente de red, el host donde residen las clases debe haberse configurado con la propiedad java.rmi.server.codebase. Esto permite al sistema RMI embeber el URL de la clase en la forma serializada de la clase.

Las clases sobre un host A que son visibles por otros host a través de la propiedad java.rmi.server.codebase de A, no pueden estar también disponibles a través del CLASSPATH de A. Si lo estuvieran, el sistema RMI no incluiría el URL de la clase cuando fuera serializada. La forma más fácil de protejerse contra ésto es poner todas las clases en un server HTTP y cargar todas las aplicaciones (clientes, servers, peers) desde allí.

Aún si un objeto serializado contiene el URL desde el cual puede cargarse la clase, un cliente o peer la cargará localmente si es que está disponible.

2.- Interfaces del cliente

Cuando se escribe un applet o una aplicación que usa objetos remotos, el programador puede necesitar conocer las interfaces de clientes visibles del sistema RMI.



2.1.- Interface Remote

La interface java.rmi.Remote sirve para identificar todos los objetos remotos; cualquier objeto que sea remoto debe implementar, directa o indirectamente, esta interface. Todas las interfaces remotas deben ser declaradas públicas.

2.2.- Clase RemoteException

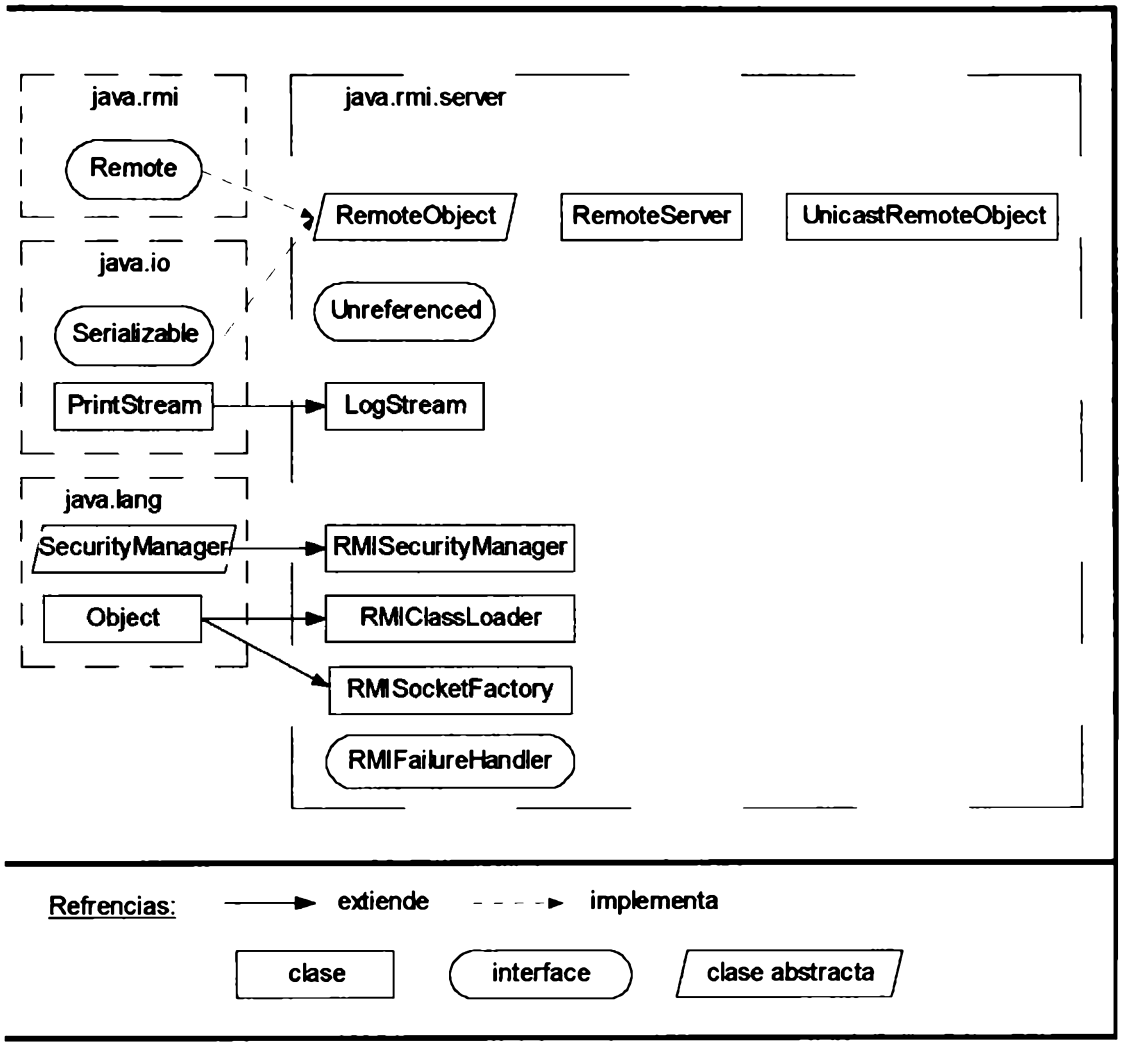
Todas las excepciones remotas son subclases de java.rmi.RemoteException. Esto permite a las interfaces manejar todos los tipos de excepciones remotas y distinguir excepciones locales y excepciones específicas del método, de aquellas tiradas por los mecanismos de objetos distribuidos subyacentes.

2.3.- Clase Naming

La clase Naming permite que los objetos remotos sean definidos y retornados usando la sintaxis familiar de URL. El URL consiste de protocolo, host, port y campos de nombre. El servicio Registry sobre un host y port especificados se usa para realizar la operación requerida.

3.- Interfaces del server

Cuando se implementa un server, las interfaces del cliente están disponibles y son extendidas con aquellas que permiten la definición, creación y exportación de objetos remotos.



3.1.- Clase RemoteObject

La clase java.rmi.server.RemoteObject implementa el comportamiento d ava.lang.Object para los objetos remotos.

3.2.- Clase RemoteServer

La clase java.rmi.server.RemoteServer es la superclase común de todas la mplementaciones server y provee el ambiente para soportar un amplio rango d semánticas de referencias remotas. En este momento, la única subclase que soporta e UnicastRemoteObject.

3.3.- Clase UnicastRemoteObject

La clase java.rmi.server.UnicastRemoteObject provee soporte para referencias objetos activos punto a punto usando streams basados en TCP. La clase implementa u objeto server remoto con las siguientes características:

- Las referencias son válidas a lo sumo durante la vida del proceso que crea objeto remoto.

- Se usa transporte basado en conecciones TCP.
- Las invocaciones, parámetros y resultados usan un protocolo de streams para comunicarse entre clientes y server.

3.3.1.- Construcción de un nuevo objeto remoto

En una máquina virtual Java que está corriendo como server, los objetos remotos definidos por el desarrollador pueden ser creados por la aplicación server. Cuando la clase de un objeto remoto extiende a UnicastRemoteObject, el constructor crea y exporta el objeto remoto. El constructor es invocado desde el constructor correspondiente a la clase del objeto remoto. El constructor por defecto crea un nuevo objeto remoto unicast usando un puerto anónimo.

3.3.2.- La exportación de una implementación que no extiende a RemoteObject

El método exportObject() se usa para exportar un objeto remoto unicast, que no fue implementado extendiendo la clase UnicastRemoteObject. El método exportObject() se llama con el objeto a ser exportado sobre un puerto anónimo. El objeto debe ser exportado antes de ser pasado por primera vez en una llamada RMI, ya sea como parámetro o como valor de retorno; de otra forma se tira la excepción java.rmi.server.StubNotFoundException.

Una vez exportado, el objeto puede ser pasado como argumento o retornado como resultado en una llamada RMI. Cuando el objeto remoto se pasa durante el almacenamiento, se realiza una busqueda a fin de encontrar el stub remoto que matchee con la implementación del objeto y luego, ese stub es pasado o retornado en su lugar.

3.4.- Interface Unreferenced

La interface java.rmi.server.Unreferenced permite al objeto server recibir la notificación de que no hay más referencias remotas hacia él. El mecanismo de recolección de residuos distribuido mantiene un conjunto de referencias remotas para cada objeto remoto. Mientras que un cliente mantenga una referencia remota, el runtime de RMI guarda una referencia local al objeto remoto. Cuando el conjunto se vacía, se invoca al método Unreferenced.unreferenced. No se requiere ninguna acción por parte de la implementación, ni el soporte de Unreferenced.

Mientras exista alguna referencia local al objeto remoto, éste puede ser pasado en llamadas remotas o retornado a los clientes. El proceso que recibe la referencia es agregado al conjunto de referencias de esa referencia. Cuando las nuevas referencias no existan más será invocado Unreferenced. De esta forma, el método unreferenced puede ser llamado más de una vez, cada vez que se vacie el conjunto. Los objetos remotos son recolectados únicamente cuando no existan más referencias, ya sean locales o aquellas mantenidas por los clientes.

3.5.- Clase RMISecurityManager

Se puede usar la clase RMISecurityManager cuando la aplicación no requiere de funciones de seguridad especializadas, pero sí necesita la protección que ésta provee.

Este manejador de seguridad deshabilita todas las funciones, excepto las definiciones de clase y el acceso a aquellas otras clases que permitan que los objetos remotos, sus argumentos y valores de retorno sean cargados según se necesite.

Si no se ha seteado ningún manejador de seguridad, se deshabilita la carga de stubs. Esto asegura que algún manejador de seguridad debe ser el responsable de la carga de stubs y clases que sean parte de la invocación de métodos remotos. Un manejador de seguridad se setea usando System.setSecurityManager.

3.6.- Clase RMIClassLoader

El runtime de RMI usa su propio cargador de clases para cargar stubs, skeletons y otras clases que ellos necesitan. Estas clases y la forma en que son usadas, soportan las propiedades de seguridad del runtime de RMI de Java. Este cargador de clases siempre carga primero las clases que están disponibles localmente. Sólo si el manejador de seguridad lo permite, se cargarán los stubs, ya sea de la máquina local o de una fuente de la red.

El cargador de clases mantiene una cache de cargadores para URLs individuales y para las clases que han sido cargadas a partir de ellos. Cuando se carga un stub o skeleton, también se cargará cualquiera de las referencias a clases que ocurran como parámetro o valor de retorno, y estarán sujetas a las mismas restricciones de seguridad.

Los procesos servers deben declararle, al runtime de RMI, la ubicación de las clases (stubs y parámetros/valores de retorno) que estarán disponibles a sus clientes. La propiedad java.rmi.server.codebase debería ser una URL desde la cual se puedan cargar las clases stub y las usadas por ellos, mediante los protocolos normales, por ejemplo, http, ftp, etc.

El java.rmi.server.RMIClassLoader es una clase utilitaria que puede ser usada por aplicaciones que cargan clases vía un URL.

3.7.- Clase RMISocketFactory

La clase abstracta java.rmi.server.RMISocketFactory provee una interface para especificar la forma en que la capa de transporte debería obtener los sockets.

3.8.- Interface RMIFailureHandler

La interface java.rmi.server.RMIFailureHandler provee un método para especificar la forma en que el runtime de RMI debería responder cuando falla la creación de sockets del server.

3.9.- Clase LogStream

La clase LogStream presenta un mecanismo para registrar errores que son de posible interés para quienes monitorean el sistema. Esta clase es usada internamente por el registro de llamadas del server.

3.10.- Compilador stub y skeleton

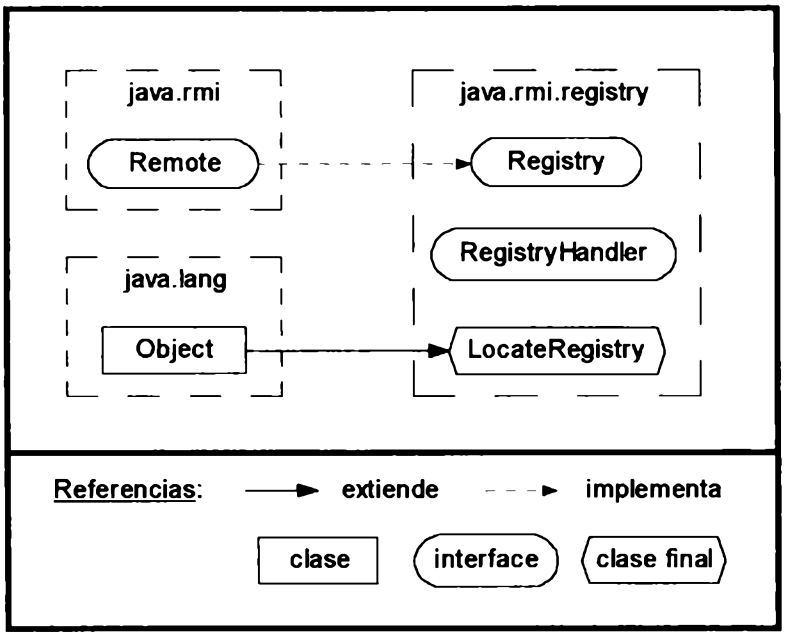
El compilador de stubs y skeletons rmic es usado para compilar los stubs y skeletons apropiados para una implementación de objeto remoto específico. El compilador es invocado con el nombre de la clase del objeto remoto calificado por el paquete al cual pertenece la clase. La clase debe haber sido compilada con exito previamente.

4.- Interfaces del registro

El sistema RMI usa la interface java.rmi.registry.Registry y la clase java.rmi.registry.LocateRegistry para proveer un servicio de arranque para la devolución y registro de objetos a través de sus nombres. Cualquier proceso server puede soportar su propio registro, o, se puede usar un registro sólo para un host.

Un Registry es un objeto remoto que mapea nombres con objetos remotos. Este puede ser usado en una máquina virtual con otras clases server o aplicaciones.

Los métodos de LocateRegistry se usan para obtener un Registry que está operando sobre un host o host y port particular.



4.1.- Interface Registry

La interface remota java.rmi.registry.Registry provee métodos para buscar, enlazar, reenlazar, desenlazar y listar los contenidos de un registro. La clase java.rmi.Naming usa la interface remota Registry para proveer un mecanismo de asociación de nombres con URLs.

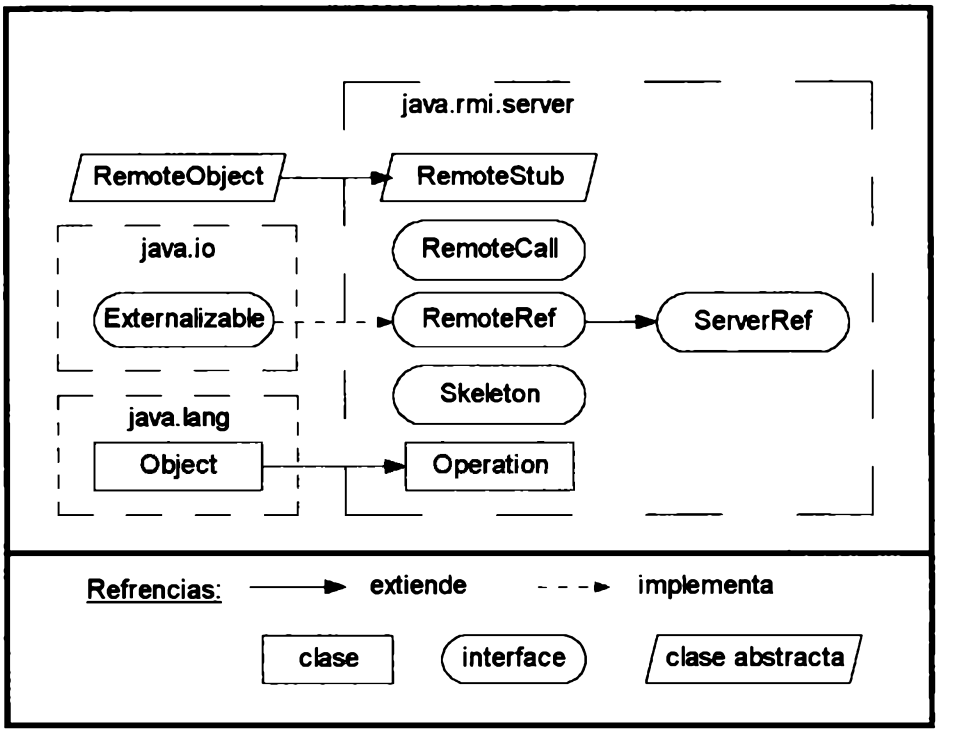
4.2.- Clase LocateRegistry

La clase java.rmi.registry.LocateRegistry contiene métodos estáticos que retornan un registry del host actual, del host actual en un puerto especificado, de un host especificado o de un puerto particular de un host especificado.

4.3.- Interface RegistryHandler

RegistryHandler sirve como interface para la implementación privada.

5.- Interfaces Stub/Skeleton



5.1.- Clase RemoteStub

La clase java.rmi.server.RemoteStub es la superclase común a todos los stubs de los clientes. Los objetos stubs son representantes que soportan exactamente el mismo conjunto de interfaces remotas definidas por la implementación actual de un objeto remoto.

5.2.- Interface RemoteCall

Es una abstracción usada por los stubs y skeletons de los objetos remotos para llevar a cabo una llamada a un objeto remoto.

5.3.- Interface RemoteRef

La interface RemoteRef representa el manejador de objetos remotos. Cada stub contiene una instancia de RemoteRef que contiene la representación concreta de una

referencia. Esta referencia remota es usada para realizar llamadas remotas al objeto remoto al cual pertenece la referencia.

5.4.- Interface ServerRef

La interface ServerRef representa el manejador, del lado del server, de la implementación de objetos remotos.

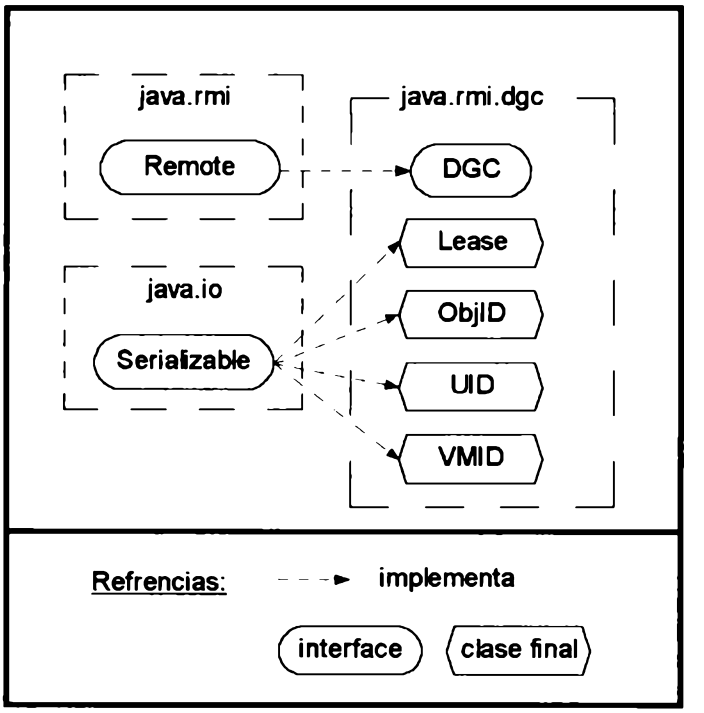
5.5.- Interface Skeleton

La interface Skeleton es usada únicamente por la implementación de los skeleton generados por el compilador rmic. Un skeleton para un objeto remoto es una entidad del lado del server que envía llamadas a la implementación del objeto remoto actual.

5.6.- Clase Operation

La clase Operation mantiene una descripción de un método Java para un objeto remoto.

6.- Interfaces del recoletor de residuos



6.1.- Interface DGC

La abstracción DGC es usada por el algoritmo de recolección de residuos distribuido del lado del server. Esta interface contiene dos métodos: dirty y clean. Se hace una llamada a dirty cuando se recupera una referencia remota en un cliente (el cliente se indica por su VMID). Se realiza una llamada a clean cuando no hay más referencias a la referencia remota en el cliente. Cuando falla una llamada a dirty, se debe realizar una llamada a clean para que se pueda guardar el número de secuencia de la

llamada, a fin de detectar futuras llamadas recibidas fuera de orden por el recolector de residuos distribuido.

Un cliente que tiene una referencia a un objeto remoto, la reserva por un cierto período de tiempo, que comienza cuando se recibe la llamada a dirty. Es responsabilidad del cliente renovar las reservas, sobre las referencias remotas que mantiene, haciendo llamadas adicionales a dirty, antes de que tales reservas expiren. Si estas expiran, el recolector de residuos distribuido asume que el objeto remoto ya no está referenciado por el cliente.

6.2.- Clase Lease

Un objeto lease contiene un identificador de máquina virtual único y una duración. Se usa para pedir y garantizar reservas a referencias de objetos remotos.

6.3.- Clase ObjID

La clase ObjID se usa para identificar objetos remotos unívocamente en una máquina virtual a través del tiempo. Cada identificador contiene un número de objeto y un identificador del espacio de direcciones, que es único con respecto a un host específico. Un identificador de un objeto se asigna a un objeto remoto cuando éste es exportado.

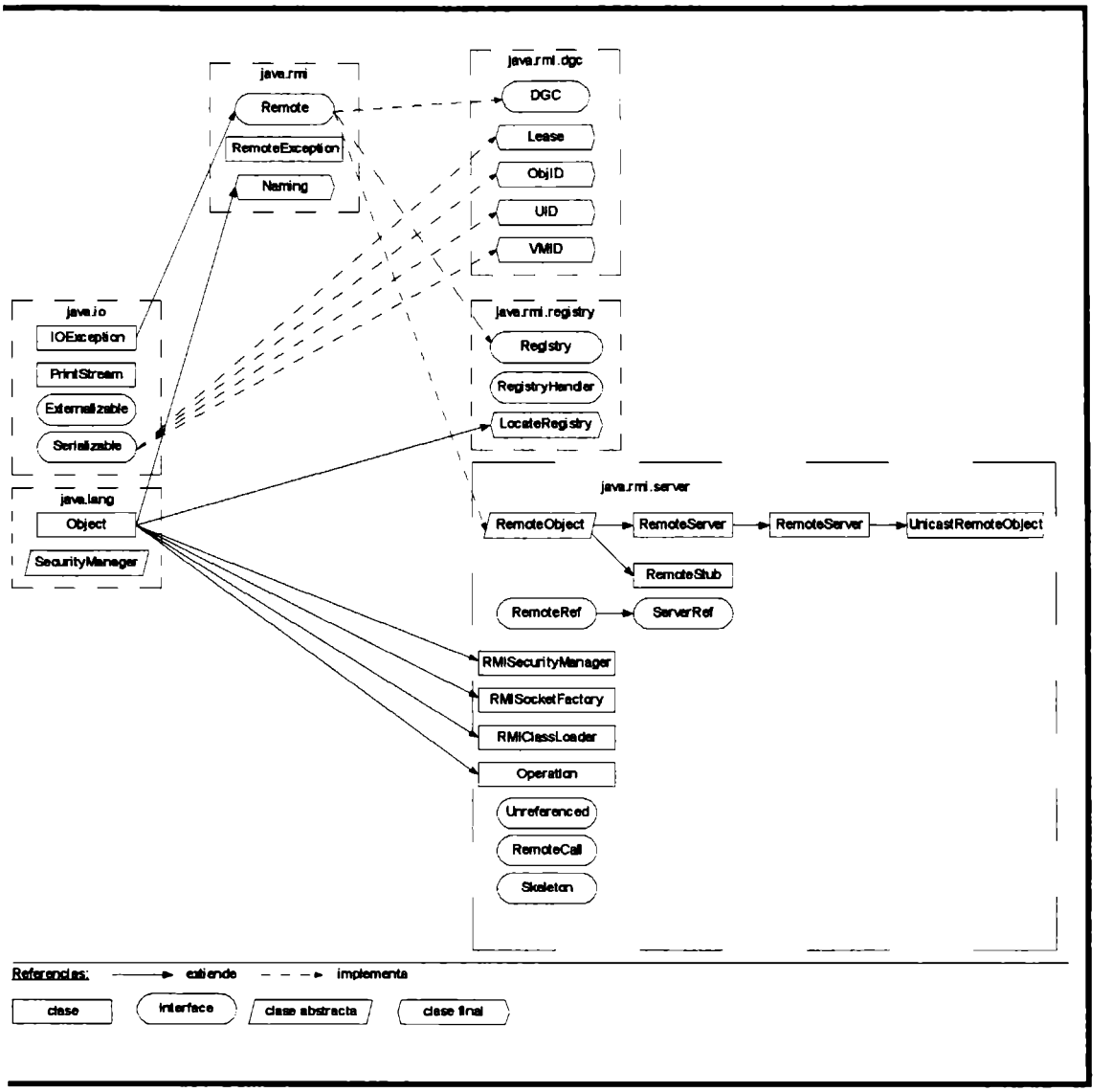
6.4.- Clase UID

La clase UID es una abstracción que permite la creación de identificadores que son únicos con respecto al host en el cual son generados. Un UID está contenido en un ObjID como un identificador del espacio de direcciones.

6.5.- Clase VMID

La clase VMID provee un identificador único universal para la máquina virtual de un cliente. Un VMID contiene un UID y una dirección de host.

7.- Gráfico general de RMI


java.rmi
Remote
RemoteException
Naming
java.rmi.dgc
DGC
Lease
ObjID
UID
VMID
java.io
IOException
PrintStream
Externalizable
Serializable
java.rmi.registry
Registry
RegistryHandler
LocateRegistry
java.lang
Object
SecurityManager
java.rmi.server
RemoteObject
RemoteServer
RemoteServer
UnicastRemoteObject
RemoteStub
RemoteRef
ServerRef
RMISecurityManager
RMISocketFactory
RMIClassLoader
Operation
Unreferenced
RemoteCall
Skeleton
Referencias:
extiende
implementa
clase
interface
clase abstracta
clase final

Apéndice C

Casos de Uso y Diseño Orientado a Objetos

1.- Casos de Uso [32]

El primer paso en la modelización de un sistema complejo debe ser la descripción del sistema, su ambiente, y la relación entre ellos. En otras palabras, se debe describir un sistema tal como se ve desde afuera, es decir, una visión de caja negra.

Los casos de uso son una forma de estructurar esta visión de caja negra. Ellos representan lo que el usuario puede hacer con el sistema. Si se identifican tempranamente todas las formas en que el sistema será usado y luego, se controla el desarrollo del mismo de manera tal de proveer esas formas, se podrá saber que se está construyendo el sistema correcto.

1.1.- Roles

Los casos de uso tienen dos roles importantes:

- *Capturar un requerimiento funcional del sistema:* Un modelo de casos de uso define el comportamiento del sistema especificando un conjunto de casos de uso. El ambiente del sistema se define describiendo los distintos usuarios, los que luego lo usarán a través de un número de casos de uso. El modelo de casos de uso no reemplaza al modelo de objetos, ya que el primero es una visión externa del sistema, mientras que el segundo es una visión interna.

- *Estructurar cada modelo de objetos en una visión manejable:* Los casos de uso son prácticos para presentar los modelos de objetos que representan sistemas complejos en distintas visiones. Se debe construir una visión para cada caso de uso y en cada una se modelizan sólo aquellos objetos que participan en dicho caso de uso. Esto significa que un modelo de objetos complejo puede verse como un conjunto de visiones de modelos de objetos, uno para cada caso. Así, se pueden encontrar todas las responsabilidades de un objeto a través de todos los casos de uso donde ese objeto tiene un rol. Cada rol significa una responsabilidad. La responsabilidad total del mismo se obtiene integrando todas sus responsabilidades.

1.2.- Modelización de Casos de Uso

Un modelo de casos de uso es un grafo con dos tipos de nodos: actor y caso de uso. El nombre de este modelo es el nombre del sistema.

Asociado a cada nodo actor hay un nombre y una clase actor. Estos nombres son únicos. Asociado a cada nodo caso de uso hay un nombre y una clase caso de uso. Estos nombres también son únicos en el alcance del sistema.

Un nodo actor tiene un arco hacia al menos un nodo caso de uso y un nodo caso de uso tiene un arco hacia al menos un nodo actor. Esos arcos son llamados arcos de comunicación.

Una instancia de un nodo actor puede crear instancias de casos de uso, esta instancia obedece a su clase. Un arco de comunicación entre un nodo actor y un nodo caso de uso significa que el estímulo puede ser enviado entre instancias de la clase actor e instancias del caso de uso, o entre clases de caso de uso. Este último caso es relevante cuando se crean nuevas instancias.

1.3.- Nodo Actor

Los actores son objetos que residen fuera del sistema modelizado, mientras que los casos de uso son objetos que residen dentro del mismo.

Los actores son los que interactúan con el sistema y son creados fuera del control de nuestras herramientas de modelización. Ellos representan a todos aquellos que necesitan intercambiar información con el sistema.

Además del concepto de actores, existe el de usuarios. La diferencia entre ellos es que los actores son instancias de una clase, mientras que los usuarios son alguna clase de recursos que implementan esas instancias. Una instancia de una clase actor existe solamente cuando el usuario hace algo con el sistema. Un mismo usuario puede además actuar como instancia de varios actores diferentes.

1.4.- Nodo Caso de Uso

El conjunto de casos de uso es la funcionalidad completa del sistema. De la misma manera que con los actores, el caso de uso es una clase que puede ser instanciada. Cuando un caso de uso es ejecutado, esto es una instancia de clase, la cual existe mientras el caso de uso está operando.

¿Qué es un buen caso de uso? Un buen caso de uso es aquel que, una vez instanciado, es una secuencia de transacciones ejecutada por un sistema, que da un resultado medible de valores para un actor particular. Los casos de uso pueden verse como objetos que representan grandes transacciones con estados que pueden ser manipulados mediante comunicaciones con ellos. La principal diferencia entre lo que generalmente pensamos como un objeto y un caso de uso, es que los objetos pueden comunicarse con otros en el mismo sistema, mientras que los casos de uso pueden comunicarse sólo con actores fuera del sistema, y no con otros casos de uso.

¿De dónde llega un caso de uso? La modelización de casos de uso está precedida por una actividad de visionamiento, en la cual los usuarios participan para encontrar los casos de uso y las interfaces relacionadas. Para esto, se observa a los usuarios en sus lugares de trabajo y se les realizan entrevistas y preguntas a fin de describir en forma episódica, diferentes escenarios de uso (instancias de casos de uso). Como un medio para entender mejor las necesidades de los usuarios, se realiza un proceso de mejora de los esquemas de las interfaces y, cuando ellos se convierten en estables y no antes, se pueden desarrollar prototipos.

¿Cómo son los distintos casos de uso de escenarios? Los casos de uso se asemejan a los escenarios de uso en el mundo orientado a objetos. Sin embargo, hay varias diferencias, tanto sintácticas como semánticas. Los escenarios normalmente significan instancias de caso de uso y no tienen un equivalente en las clases casos de uso. Los casos de uso son tratados más formalmente y descriptos en un modelo de su propiedad, así como también en interacciones entre objetos en distintos modelos de objetos. En cambio, los escenarios son generalmente descriptos sólo como interacciones entre objetos.

1.5.- Clases Caso de Uso

Una clase caso de uso puede ser modelizada como una máquina de estado. Una instancia de un caso de uso atraviesa estados de esta máquina durante su tiempo de vida. Un estado representa el potencial del caso de uso (instancia). La continuación del caso de uso a partir de su estado actual, dependerá del estímulo que reciba. Un estímulo recibido de un actor (instancia), causará que el caso de uso deje su estado actual y ejecute una transacción, la cual dependerá de la combinación estado-estímulo. La transacción incluye manipulación de atributos internos de los casos de uso y salidas a los actores, que pueden ser los que crearon el caso de uso u otros que hayan estado involucrados durante el curso del mismo. La transacción finaliza cuando el caso de uso entra a un estado (posiblemente el mismo) y espera otro estímulo de un actor.

1.6.- Aplicaciones de los Casos de Uso

- El caso de uso es una herramienta que permite modelizar un sistema desde un punto de vista externo: visión de caja negra. Esta visión es esencial para el acuerdo entre la gente que usa el sistema y la gente que lo desarrolla.

- Los casos de uso pueden ser usados para estructurar un modelo de objetos complejos en visiones manejables.

- Los casos de uso unen diferentes modelos de un sistema, por ejemplo: el modelo de casos de uso con el modelo de análisis de objetos, y éste con el modelo de diseño.

- Los casos de uso son una herramienta para organizar el trabajo de los desarrolladores, ya sea el análisis de requerimientos, el diseño, la implementación, el testeo o la operación.

- Los casos de uso son casos de testeo en el test de integración de un sistema. El test de planeamiento puede comenzar en el momento en que se identifican.

- Los casos de uso son también la fuente para imprimir el manual de usuario.

2.- Diseño Orientado a Objetos [33]

La idea central es el desarrollo de un diseño del manejo de responsabilidades, seguido por un modelo de cooperación y comunicación de objetos.

2.1.- Objetivos del proceso de diseño

Inicialmente tenemos varios objetivos al describir un proceso de diseño orientado a objetos. Ellos son:

- Estimular la exploración de alternativas, mientras se proveen guías y estrategias de diseño para improvisar decisiones iniciales.

- Diseñar una visión de alto nivel especificando cómo, los objetos claves, interactúan en sus aplicaciones, antes de completar los detalles precisos de clases individuales de objetos.

2.2.- Descripción del software desde una visión externa

Históricamente, el gap entre el lenguaje de los desarrolladores de software y el de los usuarios ha sido amplio. Los usuarios describen los sistemas sin técnica, usando términos orientados a las tareas que realizan, mientras que los desarrolladores piensan en términos de representación de datos y algoritmos de software. Esto es válido también para software orientado a objetos.

Un diseño orientado a objetos describe un modelo de objetos del mundo real, los cuales residen en el reino fisico del usuario y son modelizados en el diseño de software por un desarrollador. Sin embargo, las nociones de los usuarios de un objeto fisico o conceptual difiere de nuestra visión sintética en que nosotros modelamos en nuestro software. Todavía hay un gap semántico entre estos dos mundos.

Los desarrolladores de software necesitan técnicas prácticas para salvar el gap entre las descripciones que son significativas para los usuarios y las descripciones que contienen los detalles técnicos suficientes para construir un modelo de interacción de objetos de software. Los desarrolladores, diseñadores de interfaces, usuarios, expertos de dominio y especialistas de software, necesitan trabajar juntos para manejar los detalles de las distintas perspectivas durante el proceso de desarrollo del software entero.

Un caso de uso puede ser tratado como una especificación refinada, extensible y reusable de un sistema de requerimientos. Hemos encontrado casos de uso y otras descripciones informales útiles, al proveer un enfoque y un sentido a nuestros esfuerzos en el desarrollo del software. Los casos de uso son descripciones textuales de interacciones entre un actor y nuestro software. Ellos proveen límites y restricciones al sistema a construir. Los casos de uso, en algún sentido, forman un contrato entre los desarrolladores y los usuarios del sistema.

2.3.- Conversación entre un actor y el sistema

Es útil separar qué hace el sistema de qué comunica el usuario al software. Se quieren remarcar las acciones de los actores y las actividades o respuestas del sistema. Esto nos permite agregar más a ambas partes de esta descripción y contar, en términos de alto nivel, una conversación entre un actor y el sistema. Hay dos partes centrales para esta conversación: una descripción de pedido o entrada de un actor al sistema, y una des-

cripción de las correspondientes respuestas o acciones de alto nivel, tomadas por el sistema.

También se deben listar las alternativas para el curso principal de cada conversación y expandir sus descripciones, en el caso de que alteren el curso de la conversación. Las alternativas representan una lista razonablemente completa, pero no exhaustiva, de condiciones que el software debe detectar y manejar.

Una clave para construír una buena conversación es preservar el propósito dual de guíar a los desarrolladores, quienes crearán un modelo de diseño de objetos y recordarán los eventos e informaciones importantes transferidas entre los usuarios y el sistema. Para lograr ambos objetivos, las conversaciones deben escribirse en alto nivel. Las acciones y respuestas se podrían mantener para acortar frases verbales simples.

Las conversaciones capturan el flujo de comunicación entre los actores y el sistema. Si la naturaleza o la cantidad de información cambia significativamente, también debe cambiar la demanda de nuestro modelo de objetos. Esto implica que las respuestas de las descripciones del sistema deben contener suficientes detalles y reflejar los cambios de interacción y de diseño de interfaces, si ellas son guías correctas de nuestro diseño del modelo de objetos.

Las conversaciones y la información de soporte se pueden escribir en un procesador de textos, a mano o, en situaciones menos formales, pueden ser ubicadas directamente en las descripciones de los objetos.

Los nuevos diseñadores de objetos prefieren sistemas de respuestas más detallados, mientras que los diseñadores experimentados están de acuerdo con los detalles que pueden ser inferidos a partir de otras conversaciones o documentos diseñados. Los grupos que transicionan desde el análisis y diseño de software tradicional, formal y estructurado tienden a registrar información suplementaria para complementar los casos de uso y conversaciones básicas.

Todo este soporte de información necesita ser expresado en forma comprensible y trasladado a nuestras actividades de diseño de objetos. Se deben usar las conversaciones para guíar la actividad de diseño y, otro material, para proveer explicaciones más detalladas. Una conversación puede describir una situación concreta o una conversación abstracta más general, la que requiere remover cierta cantidad de detalles del diálogo del actor y del sistema y/o agregar restricciones.

La refactorización de las responsabilidades del modelo de objetos ocurre sólo después de tener las responsabilidades iniciales asignadas a los objetos que manejan situaciones específicas. Esta refactorización puede alterar significativamente los objetos. Este es el punto en el que se trata de aprovechar la herencia de clases, para disponer de objetos similares en alguna clasificación jerárquica, y especificar responsabilidades heredadas. El propósito de la abstracción es ayudar a construír software consistente. Este objetivo entra en conflicto con la necesidad inicial de entender cómo un usuario desea interactuar con el sistema en una situación específica.

2.4.- Objetos estereotipados

En primer lugar se debe describir una lista de objetos de diseño candidatos, luego, se debe mostrar cómo colaboran para soportar cada conversación. Los objetos en un diseño pueden ser, tanto participantes activos en muchas conversaciones, como participantes dóciles, que sólo responden cuando se les pregunta. Entre estos dos extremos hay muchos niveles de comportamiento, lo que es útil para clasificar los objetos de acuerdo a sus propósitos primarios, sus modos operandi y sus utilidades generales.

Hay varias formas de caracterizar las utilidades de los objetos:

- *Objetos de dominio o trabajo:* Modelan los aspectos necesarios de un concepto que podría ser familiar para el usuario del software que diseñamos.

- *Objetos de aplicación*: Aunque también relacionan trabajo, están limitados al alcance de una aplicación. Algunos objetos que trabajan en un contexto simple pueden ser reusados y generalizados para trabajar en numerosas aplicaciones relacionadas. Los objetos diseñados para trabajar en dicho contexto son ciertamente simples para conceptualizar. Ellos acarrean menos "equipaje" que si fueran diseñados para ser usados en varias aplicaciones, por lo que son más faciles de utilizar y mantener.

- *Objetos utilitarios:* Son objetos de aplicación no específicos. Es extremadamente útil diseñar nuevos objetos utilitarios que soporten, explícitamente, políticas de sistemas o prácticas de programación de aplicaciones comunes.

2.5.- Comportamiento de objetos estereotipados

Como punto de partida, se deben estereotipar objetos como controladores, coordinadores, estructuradores, poseedores de información, proveedores de servicios o interfaces. Esos objetos generalmente son útiles en aplicaciones específicas o en distintas aplicaciones de trabajos específicos. Cuando se diseña el estilo de control de las aplicaciones, se necesita poner atención en aquellos objetos que coordinan, inicializan o monitorean la mayoría de las actividades de las aplicaciones.

Detalles del comportamiento estereotipado:

- Los *controladores* son centros activos y vitales de influencia. A menudo esos objetos pueden ser identificados por su nombre en un diseño existente. Es común controlar objetos mediante *administradores*, *controladores*, *manejadores* o *conductores*. Esos objetos son los responsables de controlar un ciclo de acción, el cual puede ser repetitivo, con un branch lógico condicional, o ser iniciado y ejecutado sobre una detección inmediata de cierto conjunto de eventos o circunstancias.

- Los *coordinadores* son los administradores y delegadores en un sistema. A menudo requieren pares de clientes con objetos ejecutando un pedido de servicio. Un coordinador puede responder a un pedido, estableciendo brevemente un contexto apropiado, y luego delegandolo a uno o más objetos en su área de influencia.

- Los *estructuradores* mantienen relaciones entre aplicaciones de objetos. En muchas aplicaciones, el trabajo con objetos tiene relaciones estructurales muy complejas. Cuando se clasifica a un objeto de estructura primaria, se debe pensar primero, y ante todo, acerca de qué relaciones debería mantener entre los objetos que conoce, y segundo, qué responsabilidades útiles, adicionales, le son apropiadas.

- Los *controladores de información* tienen valores que pueden ser consultados. De vez en cuando pueden ser útiles para crear objetos responsables de producir información. En el lenguaje de programación procedural, se pueden declarar valores constantes. En el diseño de objetos, los *controladores de información* son un concepto equivalente.

- Los *proveedores de servicio* son diseñados para ejecutar una demanda de operación o actividad simple. Los servicios de objetos puros, a menudo son el producto de un gran diseño factorizado. Tal diseño consiste de muchas clases de objetos con un comportamiento altamente especializado. Una razón para crear objetos de servicio es facilitar las características opcionales o configurables del software. También, como la mayoría de las responsabilidades son agregadas a una clase, puede ser complejo integrar nuevas responsabilidades con las existentes. Crear un proveedor de servicios puede reducir la complejidad.

- Las *interfaces* son los límites de una aplicación orientada a objetos. Pueden ser diseñadas para soportar comunicaciones con usuarios, con otros programas o con servicios disponibles externamente. Las objetos interfaces se adaptan a distintos tamaños, formas, sabores y niveles conceptuales. Son responsables de salvar las diferencias entre el mundo sin objetos y el mundo de los objetos y mensajes. Las interfaces pueden transformar eventos o pedidos externos en mensajes pedidos por objetos de aplicaciones interesadas.

2.6.- Responsabilidades y colaboraciones

La identificación y estereotipación de clases centrales en una aplicación, es el primer paso. Luego, se deben describir las acciones a ser cumplidas y cuáles de los objetivos trabajarán juntos para satifacerlas. Una responsabilidad es un subconjunto cohesivo de comportamientos, definido por un objeto. Las responsabilidades de un objeto son declaraciones de alto nivel acerca de las acciones que éstas puedan ejecutar, así como también, el conocimiento que mantienen y proveen sobre la demanda.

Una buena forma de determinar las responsabilidades de un objeto es respondiendo estas preguntas:

- ¿Qué necesita un objeto para conocer en qué orden cumplir cada objetivo con el que está involucrado?
- ¿De qué pasos debería ser responsable el objeto, después del cumplimiento de cada objetivo?

Los objetos no existen aisladamente. Las aplicaciones orientadas a objetos de tamaño moderado constan de cientos de objetos cooperando. Una colaboración es un

Bibliografía

[1] *Groupware, Communication, Collaboration, Coordination,* Internet.

[2] M. L. Campos, L. A. Banos, M. C. Cavalcanti, *INTERCONNECT: An extensible workbench for customizing cooperative work applications,* Universidad Federal de Rio de Janeiro, Setiembre 1995.

[3] Marcelo O. Lerra, Mauricio O. Mendiburu, *ABACO: Un Ambiente Basado en Artefactos para Trabajos Colaborativos,* Marzo 1994.

[4] *Collaborations using Desktop Videoconferencing,* Internet.

[5] Sara Latta, *Building Collaborations,* Internet.

[6] Richard T. Kouzes, *Collaboratories: Scientists Working Together Apart,* Enviromental Molecular Sciences Laboratory, Pacific Nortwest Laboratory.

[7] Donald A. Norman, *CSCW,* Communications of the ACM, Diciembre 1991.

[8] Donald A. Norman, *Collaborative Computing: Collaboration first, computing second,* Communications of the ACM, Diciembre 1991.

[9] Mike Dilworth, *The forum of Computer Supported Collaborative Working Collaboration using the World Wide Web,* Internet, Diciembre 1995.

[10] Donald A. Norman, James C. Spohrer, *Learner Centered Education,* Communications Learner Centered Education.

[11] Marion A. Barfurth, *Understanding the Collaborative Learning Process in a Tecnology Rick Enviroment: the case of children's desaagreement,* Departamento de Ciencias de la Educación de Québec a Hull, 1994.

[12] Steward Nelson, Novell, Groupware Division, *Collaborative Computing: Empoweing People to Accelerate the Decision Making Process,* Internet.

[13] *Collaborative Computing,* Internet.

[14] Jeff Shapiro, *Collaborative Computing. Multimedia Across the Network,* AP Profesional, 1996.

[15 *CSCW (Computer Supported Collaborative Work),* Internet.

[16] Jonathan Grudin, *CSCW: History and focus,* Information and Computer Science Department, University of California, Irvine.

[17] David Coleman, *Groupware: Tecnology and Applications, an Overview of Groupware,* Internet.

[18] *Groupware,* Internet.

[19] Ola Berge, *On the Beaten Path,* Internet, Enero 1996.

[20] *Shared decision making: a definition,* Internet, Setiembre 1995.

[21] Jack Rehder, *Working in Small Groups.*

[22] Ryan K. Lahti, *Group Decision Making within the organization,* University of North Texas, 1996.

[23] E. Krol, E. Hoffman, *What is the Internet?,* University of Illinois - Merit Network Inc , Mayo 1993.

[24] David Flanagan, *Java in a Nutshell. A Desktop Quich Reference for Java Programmers,* O'Reilly & Associates, Inc, Febrero 1996.

[25] James Gosling, Henry McGilton, *The Java Language Enviroment: A white paper,* Internet.

[26] *The Java Language Specification,* Internet.

[27] Frank Yelten, *Low Level Security in Java,* Sun Microsystems, Java Products Group, 1996.

[28] James Gosling, Frank Yelten, *The Java Team Note,* 1996.

[29] Petter Fingar, Jim Stikeleather, *Distributed Objects for Bussiness. Getting Started with the next Generation of Computing,* 1996.

[30] *RMI,* Sun Microsystems, Inc, 1996.

[31] Rob Kling, *Cooperation, Coordination and Control in Computer - Supported Work,* Communications of the ACM, Diciembre 1991.

[32] Ivar Jacobson, *The Use-Case Construct in Object-Oriented Software Engineering.*

[33] Rebecca Wirfs - Brock, *Designing Objects and Their Interaction: A Brief Look at Responsability-Driven Design,* Digitalk, Inc.