TRABAJO DE GRADO

Protocolo de Autenticación y Autorización Kerberos

Autoras

González Eliana - Riccio Sandra

Director

Lic. Díaz Javier

Indice

Capítulo 4: Lenguaje JAVA

Capítulo 5: JDBC (Java Database Conectivity)

Capítulo 6: RMI (Método de Invocación Remota)

Introducción

Nuestra Tesis se basa en estudiar e implementar el Protocolo de Autenticación y Autorización Kerberos utilizado para otorgar seguridad a servicios sobre una red.

Podemos decir que seguridad de red y seguridad de información se refieren a la confianza en que la información y los servicios disponibles en la red no puedan ser accedidos por usuarios no autorizados. Seguridad implica confianza, incluyendo la integridad de los datos, la prevención de accesos no autorizados a los recursos computacionales, que los recursos estén libres de intromisiones y que los recursos se encuentren libres de interrupciones en el servicio.

En términos generales los problemas de seguridad en la red y los mecanismos de software que ayudan a que la comunicación en la red sea segura, se pueden dividir en tres conjuntos:

- *Autenticación*: verificar la identidad del usuario.
- *Autorización*: determinar si el usuarios está habilitado para utilizar el servicio.
- *Integridad*: proteger la información de cambios no autorizados.

Con la aparición de Internet como un medio de comunicación masivo y su gran crecimiento día tras día, es imprescindible otorgar algún mecanismo de seguridad que garantice la integridad y la privacía de los datos. Así, surgió la posibilidad de implementar el mecanismo del protocolo de Kerberos que será utilizado directamente por aplicaciones específicas (desarrollado por el proyecto Athena del Instituto Tecnológico de Massachusetts) para ayudarnos a solucionar los problemas de seguridad mencionados.

Kerberos es una conexión de software que se emplea en una red grande para establecer la identidad declarada de un usuario. Utiliza una combinación de encriptación (referenciado en el Capítulo 2 de la tesis) y Bases de Datos distribuidas de tal forma que un usuario pueda registrarse y comenzar una sesión desde cualquier computadora localizada en la red mediante la obtención de tickets para servicios de un servidor especial conocido como TGS (servidor despachador de tickets); cada ticket contiene información para identificar al usuario o servicio encriptada con la clave privada para el servicio. Como sólo Kerberos y el servicio conocen dicha clave, se considera que el mensaje está genuinamente originado en la fuente y que no fue adulterado en el transporte del mismo. El ticket otorgado por el TGS contiene una nueva clave de sesión que solo conoce el cliente y el servicio afectado. Esta clave será utilizada para encriptar las transacciones que ocurren durante la sesión.

Una de las ventajas es que el ticket tiene un tiempo de vida específico, y una vez que éste expira, debe solicitarse un nuevo ticket al TGS para poder seguir utilizando el servicio. Para cada servicio se requiere un ticket distinto. Otra ventaja es que el usuario no debe reingresar la password cada vez que requiere un servicio, porque si el ticket TGS no expiró puede reusarlo para pedir otro ticket de servicio deseado. Por este motivo, el tiempo de vida del ticket TGS deberá ser mayor que el tiempo de vida del ticket de servicio (explicado en el Capítulo 3).

Este protocolo otorga seguridad a usuarios registrados en la red, o sea, usa una autenticación muy fuerte.

Hemos implementado el Protocolo de Autenticación y Autorización Kerberos usando el lenguaje de programación Java (referenciado en el Capítulo 4), que podrá ser utilizado por aplicaciones kerberizadas, distribuidas, fuertemente autenticadas, ya que cuando un usuario quiere acceder a un servicio, el servidor Kerberos debe verificar la existencia de la identificación del usuario en la Base de Datos. Elegimos Java como lenguaje de programación porque es simple, orientado a objetos, portable, interpretado, dinámico, seguro y distribuido. Nos enfocamos principalmente al servicio RMI que permite hacer invocaciones a métodos remotos (explicado en el Capítulo 6), como también a JDBC y SQL para realizar los accesos a la Base de Datos (explicado en el Capítulo 5).

Nuestra implementación del protocolo Kerberos está compuesta por tres paquetes:

- *Kerberos,* que implementa el protocolo propiamente dicho
- *Encriptación,* que implementa el método DES (referenciado en el capítulo 2) utilizado para encriptar los tickets y autenticadores
- *Clientes,* que implementa una sencilla aplicación kerberizada que usa el protocolo Kerberos para otorgar seguridad.

Queremos destacar que el desarrollo de nuestra tesis no es una simulación del protocolo Kerberos sino una implementación del mismo en una arquitectura distribuida. Para realizarla tuvimos que introducirnos al lenguaje de programación Java. Al interiorizarnos en el aprendizaje del lenguaje, nos dimos cuenta que los accesos a métodos remotos se podían hacer mediante RMI (Invocación al Método Remoto), el cual es difícil de entender, pero permite en forma eficiente realizar los accesos remotos.

Java permite acceder a múltiples Bases de Datos, por este motivo estudiamos los accesos a través de los drivers JDBC y ODBC correspondientes a las Bases de Datos Oracle 7 y SQL*Server, respectivamente.

La versión 4 del Protocolo Kerberos utiliza el algoritmo DES (Data Encryptation Standard) para encriptar y desencriptar los datos, por tal motivo tuvimos que entenderlo e implementarlo en el lenguaje mencionado.

Los temas anteriormente detallados no fueron vistos en el transcurso de nuestra carrera y sólo algunos de ellos recién hoy en día están siendo incorporados.

Agradecimientos

Nos gustaría agradecer a los integrantes del LINTI por habernos permitido utilizar el hardware necesario para realizar las pruebas de implementación, en particular a la Lic. Laura Fava , quien nos brindó su ayuda para resolver los inconvenientes que nos fueron surgiendo en el lenguaje Java durante dicha implementación.

Así mismo, queremos agredecer al Ing. Manuel Jerez quien nos ayudó a realizar las conexiones de red e instalaciones de la Base de Datos Oracle 7.

Capítulo 1

Seguridad

1.1.- Término Seguridad

Las computadoras y las redes de datos necesitan ciertas precauciones para ayudar a mantener segura la información, porque a menos que estén en un lugar cerrado, con acceso controlado y no tenga conexiones desde afuera, las computadoras están en riesgo.

La seguridad en un ambiente de redes es importante debido a que se guarda información de gran valor en las computadoras y se tiende a garantizar que esta información sea íntegra y segura. Esta información no debe ser accedida por ningún impostor ya sea para su uso o para modificarla.

Los términos Seguridad de Red y Seguridad de información se refieren, en sentido amplio, a la confianza de que la información y los servicios disponibles en una red no puedan ser accedidos por usuarios no autorizados. Seguridad implica confianza en el hecho que no existan accesos no autorizados a los recursos, que los recursos estén libres de intromisiones, y que garantice la integridad de los datos.

Proporcionar seguridad para la información requiere protección tanto para los recursos físicos como para los abstractos. Los recursos físicos incluyen dispositivos de almacenamiento como cintas magnéticas y discos hasta cables, puentes y ruteadores que comprenden la infraestructura de la red. Los recursos abstractos, como la información, son más difíciles de proteger que los recursos físicos debido a que la información es fugaz.

La seguridad debe proteger la integridad de los datos así como también su disponibilidad.

1.2.- Análisis de los Niveles de Seguridad

Se usan varios niveles de seguridad para proteger de los ataques al hardware, al software y a la información guardada.

Dichos niveles describen diferentes tipos de seguridad física, autenticación del usuario, confiabilidad del software tanto del sistema operativo como de las aplicaciones del usuario.

- *Nivel D1:* es la forma más elemental de seguridad. Este estándar parte de la base que todo el sistema no es confiable. No hay protección disponible para el hardware, el sistema operativo se compromete con facilidad y no hay autenticación con respecto a los usuarios. Por ejemplo, MS-DOS, Ms-Windows carecen de un sistema definido para determinar quién trabaja en el teclado y no tienen control sobre la información que puede introducirse en los discos rígidos.
- *Nivel C1:* sistema de protección de seguridad discrecional, describe la seguridad disponible en un sistema típico Unix. Existe algún nivel de protección para el hardware, los usuarios deberán identificarse a sí mismos

por medio de un nombre de usuario y una contraseña para determinar qué derechos de acceso a la información tiene cada usuario.

- *Nivel C2:* Incluye características de seguridad adicional que crean un medio de acceso controlado. Este medio tiene la capacidad de reforzar las restricciones a los usuarios basados en niveles de autorización. Además, este nivel de seguridad requiere auditorías del sistema incluyendo la creación de un registro de auditoría para cada evento que ocurre en el sistema.
- *Nivel B1:* protección de seguridad etiquetada. Es el primer nivel que soporta seguridad de multinivel, como la secreta y la ultra secreta. Este nivel parte del principio que un objeto bajo control de acceso obligatorio no puede aceptar cambios en los permisos hechos por el dueño del archivo.
- *Nivel B2:* protección estructurada, requiere que se etiquete cada objeto. Este es el primer nivel que se refiere al problema de un objeto a un nivel más elevado de seguridad comunicado con otro objeto a un nivel inferior.
- *Nivel B3:* nivel de dominios de seguridad, refuerza a los dominios con la instalación de hardware.
- *Nivel A:* nivel de diseño verificado. Es hasta el momento el nivel más elevado de seguridad incluyendo un proceso exhaustivo de diseño, control y verificación.

1.3.- La seguridad de la información en las redes

En nuestro país, tanto en el mundo empresarial como en el institucional no existe una cultura de la seguridad, pero sí en los países de nuestro entorno económico y cultural.

Si no se comprende realmente la necesidad de la seguridad, difícilmente se aceptará, salvo en su aspecto físico que es el más visible, ya que es cara y es una partida fácil de eliminar de los presupuestos cuando hay que economizar costos.

Desgraciadamente, los efectos de una falta de seguridad, sólo se descubren cuando ya es demasiado tarde, cuando ya ha ocurrido el percance, ya sea de tipo físico o lógico.

El empleo de redes de comunicaciones obliga a implantar medidas de seguridad. La seguridad de la información en las redes debe comprender:

- *Confidencialidad:* garantizando que los datos transmitidos sólo sean conocidos por aquellos que estén autorizados a ello y en la medida que previamente se haya previsto.
- *Integridad:* asegurando que cualquier alteración, por mínima que sea, será detectada.
- *Disponibilidad:* facilitando que los datos estén accesibles cuando sean necesarios para las personas autorizadas.
- *Autentificación:* acreditando que el remitente del mensaje es quien dice ser y no otra persona.

1.4.- Problemas de Seguridad

Los problemas de seguridad se pueden dividir en tres aspectos:

- *Acceso al servidor*

Este es el primer muro con el que cualquier intruso se v encontrar. Es lógico pensar que debe ser lo más consistente posible.

Nuestro servidor tendrá dos tipos de usuarios: internos y remotos. De los primeros no nos ocuparemos, aunque en cualquier plan de seguridad serio deben tenerse muy en cuenta. De momento, nos obsesionaremos con la amenaza de los usuarios remotos.

Una de las formas de proteger al servidor de intrusos remotos es implantar un buen plan de contraseñas. Un plan de contraseñas debe especificar como mínimo los siguientes parámetros:

- Longitud mínima de las contraseñas. Deben tener un mínimo de 8 caracteres. Si ponemos una contraseña inferior, será factible intentar un ataque bruto.
- Caducidad de las contraseñas. Debemos poner caducidad a las contraseñas. Este parámetro limitará enormemente el posible ataque.
- Plazos de reutilización de contraseñas. No se debe permitir a un usuario reutilizar sus contraseñas en un plazo razonable. Si permitimos que las reutilicen estamos disminuyendo el valor del parámetro anterior.
- Composición de las contraseñas. Las contraseñas NO deben ser una palabra que se encuentre en un diccionario (contraseña muy débil). Una recomendación es que contengan caracteres alfanuméricos y signos de puntuación mezclados o como mínimo mayúsculas y minúsculas mezcladas.

- *Protección de la información*

Ahora supongamos que por algún error o por la astucia del intruso, éste ha conseguido entrar en nuestro servidor. Esto quiere decir que el primer muro lo ha logrado traspasar. Debemos ponerle todo tipo de trabas, para hacerle desgraciada su visita y que no la vuelva a repetir.

Resumiremos este aspecto en dos partes:

- Protección de la información del sistema

El sistema operativo posee información vital que se debe proteger. Esta información puede darle al intruso nuevas formas de penetrar en nuestro servidor o de introducirse en otros sistemas conectados en red.

La manera de proteger esta información dependerá del nivel de acceso que un intruso haya logrado. Normalmente todos los sistemas operativos tienen una cuenta con acceso a todos los recursos, denominada Administrador o root. Dicha cuenta posee un control absoluto sobre el sistema y, si el intruso ha conseguido entrar con los derechos del Administrador, tenemos serios problemas. Esta cuenta es la que mejor hay que proteger. La información del sistema esta guardada en archivos del disco duro del servidor. Dicha información no la podemos encriptar puesto que el propio sistema la usa. Las nuevas versiones de los sistemas operativos ya vienen con este tema un poco resuelto, aunque no del todo.

Así, debemos proteger los ficheros de claves de acceso del sistema, los ficheros de control del sistema y los programas que ofrecen los servicios. Si el intruso tiene acceso a los programas del sistema puede sustituir alguno por un Caballo de Troya y asegurarse la entrada futura al sistema sin que nos hayamos dado cuenta. Si tiene acceso al fichero de claves de acceso, no hace falta decir que puede pasarnos.
En definitiva, debe limitar el acceso a los archivos del sistema operativo.

- Protección de la información de la empresa

Debe ver cual es la información confidencial en su empresa y para quién. Debe tener cuidado a la hora de almacenar dicha información. Por ejemplo, no es una buena idea almacenar la información más sensible de su empresa en el servidor conectado a Internet.
Para proteger su información debe analizar cuales son los requisitos de acceso necesarios para los usuarios remotos. Si requiere mayor protección, debe cifrar dicha información con algoritmos de cifrado lo más robustos posibles.

- *Seguimiento de actividades*

Esta es la tercera parte del problema de la seguridad. Debemos vigilar las actividades que se desarrollan en nuestro servidor.
Mediante esta vigilancia podemos comprobar entre otras cosas lo siguiente:

- Accesos repetitivos.
- Accesos a recursos no permitidos.
- Uso fuera de horario de un usuario autorizado.
- Cambios en archivos del sistema sospechosos.

Como se puede ver, esto es una pequeña muestra de la valiosa información que nos aportará un buen seguimiento de actividades.
Monitorizando el sistema, podremos bloquear a un intruso antes de que haga cualquier maldad.

Debemos entender las dos primeras partes como muros que se ponen a los intrusos y la última, como la vigilancia que se debe realizar de los citados muros.
El objetivo es poner el mayor número de trabas posibles y de obtener una gran cantidad de información sobre las actividades que se desarrollan en nuestro servidor.

1.5.- Políticas de Seguridad

Antes que una organización implante un proyecto de seguridad de red, debe asumir riesgos y desarrollar una política clara, considerando los accesos y protección de la información. Las políticas necesitan especificar quiénes tendrán garantizado el acceso a cada parte de la información, deben establecerse las reglas individuales a seguir y las formas en que la organización reaccionará ante las transgresiones.
Establecer una política de información y educación a los empleados es muy importante, ya que las personas son, por lo general, el punto más susceptible de cualquier esquema de seguridad. Un trabajador malicioso, descuidado o ignorante de las políticas de información de la organización puede comprometer la seguridad.

Una política de información debe ser lo suficientemente amplia para cubrir información representada en papel y en una computadora, y debe dirigirse a aspectos como la información entrante así como la que sale de la organización.

¿Cómo diseñar una política de seguridad de red?

Una política de red es un documento que describe los temas de seguridad de red de una organización y se convierte en el primer paso para construir barreras de protección efectivas.

Dicha política de red justifica su uso si vale la pena proteger los recursos e información que tiene la organización en las redes. Debe tener presente que la política de red no disminuirá la capacidad de la organización.

Una política de seguridad de red efectiva es algo que todos los usuarios y administradores deberán aceptar.

Definirla significa desarrollar procedimientos y planes que salvaguarden los recursos de la red contra pérdidas y daños. Debe tener en cuenta:

- ¿Qué recursos se quieren proteger?
- ¿De quién necesita protegerlos?
- ¿Qué tan reales son las amenazas?
- ¿Cuál es la importancia del recurso?

El costo de proteger las redes de una amenaza debe ser menor que el costo de la recuperación, si es que se ve afectado por la amenaza de seguridad.

1.6.- Análisis de Riesgos

La razón para crear una política de red es asegurar que los esfuerzos invertidos en la seguridad son costeables. Esto significa que se deben entender qué recursos de la red valen la pena proteger y que algunos recursos son más importantes que otros.

Dicho análisis implica determinar lo siguiente:

- ¿Qué necesita proteger?
- ¿De quién debe protegerlo?
- ¿Cómo debe protegerlo?

Los riesgos se clasifican por el nivel de importancia y por la severidad de la pérdida.

Al realizar un análisis de riesgo debe identificar todos los recursos cuya seguridad está en riesgo de ser quebrantada. Recursos como hardware se encuentran en esta lista, pero los recursos como las personas que utilizan los sistemas con frecuencia se ignoran.

Una vez identificados los recursos que necesitan protección deberá identificar cuáles son las amenazas a tales recursos. Sólo se permite el acceso a los recursos de la red a los usuarios autorizados. Una amenaza común es el acceso no autorizado a los datos.

Una política de seguridad y su implantación deben evitar obstruir el sistema porque si es demasiado restrictiva o no está bien explicada, es muy posible que sea violada. [6]

1.7.- Seguridad en Internet e Intranet

Uno de los temas que es inevitable comentar en la actualidad al hablar de Internet y de la posibilidad de realizar negocios en la red, es el tema de la seguridad. El conocimiento popular indica que la Internet es "insegura".

Hace 20 años atrás, Internet comenzó como un experimento y ha cobrado hoy en día gran dimensión. Se usa como un recurso global que conecta a millones de usuarios. La demanda comenzó a crecer cuando comenzaron a conectarse a Internet los colegios, las Universidades, las empresas y los ciudadanos comunes.

Internet no es una empresa, prácticamente no existe nadie ante quien reclamar, es un conjunto de cientos de miles de redes, con una arquitectura cliente-servidor, soporte multimedia, interface amigable para no expertos y una cobertura mundial de millones de usuarios.

Su influencia para la rápida implantación de las Nuevas Tecnologías de la Información en todo el mundo viene siendo espectacular.

En Internet circula un gran volumen de información que puede ser muy importante, esto hace que cada vez más deban solucionarse los problemas de seguridad. A pesar de que en su origen la seguridad no era una de sus principales características ya que fue diseñada para resultar lo más cómoda posible a sus usuarios, no siendo condición necesaria su seguridad, hoy debido a su gran crecimiento esta característica se hace casi impresindible.

A medida que la información viaja a través de Internet, cualquier ordenador intermediario podría acceder a la información y efectuar copias. Un ordenador intermediario podría incluso intercambiar información con otro, tomando la identidad del destinatario original. Todo ello hace que la transferencia de información confidencial, como por ejemplo, contraseñas y números de tarjetas de crédito, sea susceptible al abuso.

La incorporación de una conexión a Internet en modalidad dedicada, hace necesario implementar un sistema de seguridad en la red interna de la empresa, de manera de lograr una efectiva protección de la información del usuario, la cual puede sufrir daños a través de visitas inesperadas.

Por lo importante que es el tema de seguridad hoy en día investigamos un mecanismo de seguridad para redes distribuídas con usuarios fuertemente autenticados, llamado Protocolo de Autenticación y Autorización Kerberos.

Intranet es una red de servicios internos para las empresas o las instituciones, tanto públicas como privadas, basada en los mismos estándares que Internet, donde también es fundamental contar con una política de seguridad.

Una red de estas características permite a su propietario:

- Información disponible cuando la necesite
- Actualización contínua de esa información
- Existencia de una sola fuente.

Intranet se puede implementar sobre redes propias o sobre Internet. La problemática de la seguridad estará siempre presenta pero aun más en el segundo caso.

1.8.- Resolución a los problemas de seguridad en Internet

Los problemas de seguridad en Internet y los mecanismos de software que ayudan a que la comunicación sea segura se pueden dividir en tres conjuntos:

- Problemas de autorización, autenticación e integridad.
- Problemas de privacía.
- Problemas de disponibilidad mediante el control de acceso.

Los mecanismos de autenticación resuelven el problema de verificar la identificación del cliente ya que cada una de las partes debe estar segura de que el mensaje que recibe ha sido emitido por la otra parte.

La autenticación tiene dos funciones: identificar al autor del mensaje y verificar que dicho autor se obliga legalmente con el mismo.

Una forma débil de autenticación en la red es la que utiliza direcciones IP, ya que un administrador configura una lista con direcciones IP válidas, el servidor examina la dirección IP entrante y solo acepta las que provienen de los clientes que están en la lista autorizada.

La autenticación por medio de direcciones IP es débil, porque en una red de redes en la que los datos pasan a través de ruteadores y redes intermedias, la autenticación de fuente puede ser atacada desde una de las máquinas intermedias. Por ejemplo si un impostor logra controlar un ruteador R localizado entre un cliente válido y un servidor. Para acceder al servidor, el impostor primero altera las rutas en R para dirigir el tráfico hacia él, para luego generar una solicitud utilizando la dirección IP de un cliente autorizado. Podemos concluir entonces que un esquema de autorización que utiliza una dirección IP de la máquina remota para autenticar su identidad no puede evitar ataques de parte de impostores a través de una red de redes poco segura pues un impostor que logra el control de un ruteador intermedio puede hacer las veces de un cliente autorizado.

Existen varias clases de técnicas de autenticación: el código secreto, la criptografía y el reconocimiento de caracteres físicos a larga distancia.

Estas técnicas operan como un procedimiento de verificación mediante las correspondientes comprobaciones.

El código secreto suele consistir en una combinación de números o de estos y/o letras, que en principio, sólo es conocido por su propietario. A menudo se combina con la utilización de una tarjeta magnética.

La Criptografía consiste en un sistema de cifrado de un texto con una clave confidencial y unos algoritmos de tal forma que un tercero que no disponga de la clave decodificadora no puede leer el mensaje.

La autenticación también se puede realizar a través de sistemas que permiten el reconocimiento de caracteres físicos (huella, iris, facciones, etc) a larga distancia.

Estos sistemas sólo permiten identificar a la persona, pero no conocer su voluntad. [2]

1.9.- Criptografía

Mediante la criptografía se hace incomprensible a todos un mensaje, a no ser que se conozca una clave secreta y se disponga del algoritmo apropiado.

La Criptografía es una rama del conocimiento antigua, se utilizó durante las guerras entre espartanos y atenienses y el sistema de cifrado César, utilizado por los romanos aún se usa en la actualidad.

La Criptografía lo que ofrece es seguridad, pero ésta, como todos sabemos, tiene un precio.

Aparte de necesitar las herramientas adecuadas, los mensajes habrá que cifrarlos, lo que implica normalmente aumentar su longitud y su tiempo de transmisión. En el destino será preciso descifrarlos lo que también tiene su costo.

En la Criptografía moderna existen dos tipos de criptosistemas:

- *Simétrico o de clave privada*, donde se utiliza una sola clave que sirve tanto para el cifrado como para el descifrado, y
- *Asimétrico o de clave pública*, donde cada usuario tiene dos claves: una pública para el cifrado y otra privada para el descifrado. Con la clave pública se cifra y con la clave privada se descifra lo que el otro ha cifrado con la pública.

La posibilidad de utilizar criptosistemas de clave pública permite la implantación de la firma digital. Cada usuario tiene un par de claves: una pública para el cifrado y otra secreta para el descifrado. La acción combinada de la clave pública y la secreta proporciona la confidencialidad y la autenticidad de la procedencia.

La clave secreta propia se utiliza para descifrar lo que se recibe y firmar lo que se envía. La pública permite cifrar de forma personalizada lo que se envía y comprobar quién ha firmado lo que se recibe.

De cualquier forma, el criptosistema está construído de manera que cualquier usuario puede cifrar un mensaje con su clave privada y que cualquier otro usuario puede recuperarlo con la clave pública del primero.

Ejemplo de criptosistema de clave privada es el DES (Data Encryption Standard) (explicado en el Capítulo 2) y de clave pública el RSA (Rives, Shamir y Adleman).

El uso de sistemas de clave pública puede garantizar: la autenticación, la integridad y la confidencialidad de los datos.

Integridad de la información se refiere al proceso que verifica que la información enviada esté completa y sin cambios desde la última vez que se verificó. La integridad de los datos es importante ya que la información modificada podrá crear muchos conflictos. Una manera de asegurarse que la información no ha sido cambiada es mediante las sumas de verificación. Cualquier método de encriptación también ofrece integridad ya que el oponente primero debería desencriptar el mensaje antes de modificarlo.

El cifrado de clave pública también puede manejar problemas de privacía. Por ejemplo si un emisor y un receptor utilizan este tipo de esquema, el emisor puede garantizar que solo el receptor pueda leer el mensaje, encriptando el mismo con la clave pública del receptor, luego el receptor utiliza su clave privada para descencriptarlo. Dado que solo el receptor involucrado tiene la clave privada necesaria ninguna otra parte puede decodificar el mensaje.

Control de acceso a Internet o muros de seguridad: a diferencia de los mecanismos de autenticación y privacía que se pueden añadir a los programas de aplicación el control de acceso a la red de redes requiere cambios en los componentes básicos de la misma, teniendo en cuenta una combinación cuidadosa de restricciones en la topología de la red, en el almacenamiento intermedio de la información y en el

filtrado de paquetes. Una sola técnica ha emergido como la base para el control de acceso a Internet instalando un bloque conocido como muro de seguridad en la entrada de la red que será protegida. Un muro de seguridad divide una red de redes en dos regiones protegiendo a los ruteadores, las computadoras y los datos de la red interna de una organización contra comunicaciones indeseables que provengan del exterior. Cuando la red de redes tiene más de una conexión externa deberá contar con un muro de seguridad en cada entrada, con idénticas restricciones de acceso. Entonces un muro de seguridad bloquea todas las comunicaciones no autorizadas entre las computadoras de la organización y computadoras de organizaciones externas.

La seguridad en la red está dando lugar a grandes esfuerzos para la prevención de todo tipo de problemas tanto externos como internos que puedan afectar al normal funcionamiento de las empresas.

Así pues los sistemas de seguridad en Internet se preocupan por desarrollar fórmulas efectivas de seguridad en el acceso y en las transacciones.

No debemos olvidar que al conectarnos a una red lo que estamos haciendo es abrir una puerta en nuestro sistema de información. Si no protegemos esa puerta de forma adecuada por la misma pueden entrar tanto amigos como enemigos por lo que debemos establecer el filtro adecuado para que solo puedan entrar amigos. No obstante, debemos tener en cuenta, que si establecemos filtros muy rigurosos la velocidad de la transmisión puede resentirse llegando en casos extremos a paralizar la misma. [3]

Capítulo 2

Mecanismos de Encriptación

2.1.- Encriptación

La encriptación puede utilizarse para proteger a los datos en tránsito así como los datos guardados. Algunos vendedores ofrecen dispositivos de encriptación de hardware que pueden emplearse para encriptar y desencriptar datos en conexiones punto a punto.

La encriptación puede definirse como el proceso de tomar información que existe de manera legible y convertirla en una forma que otros no pueden entender.

Si el receptor de los datos encriptados desea leer los datos originales , éste deberá convertirlos al original mediante un proceso llamado desencriptación. La desencriptación es el proceso inverso de la encriptación. Para realizar la desencriptación el receptor deberá tener una clave, la cual es necesario guardarla y distribuirla con cuidado.

La ventaja de usar la encriptación es que , aunque otros métodos para proteger sus datos fueran vencidos por un impostor, los datos todavía carecerán de significado para él. [6]

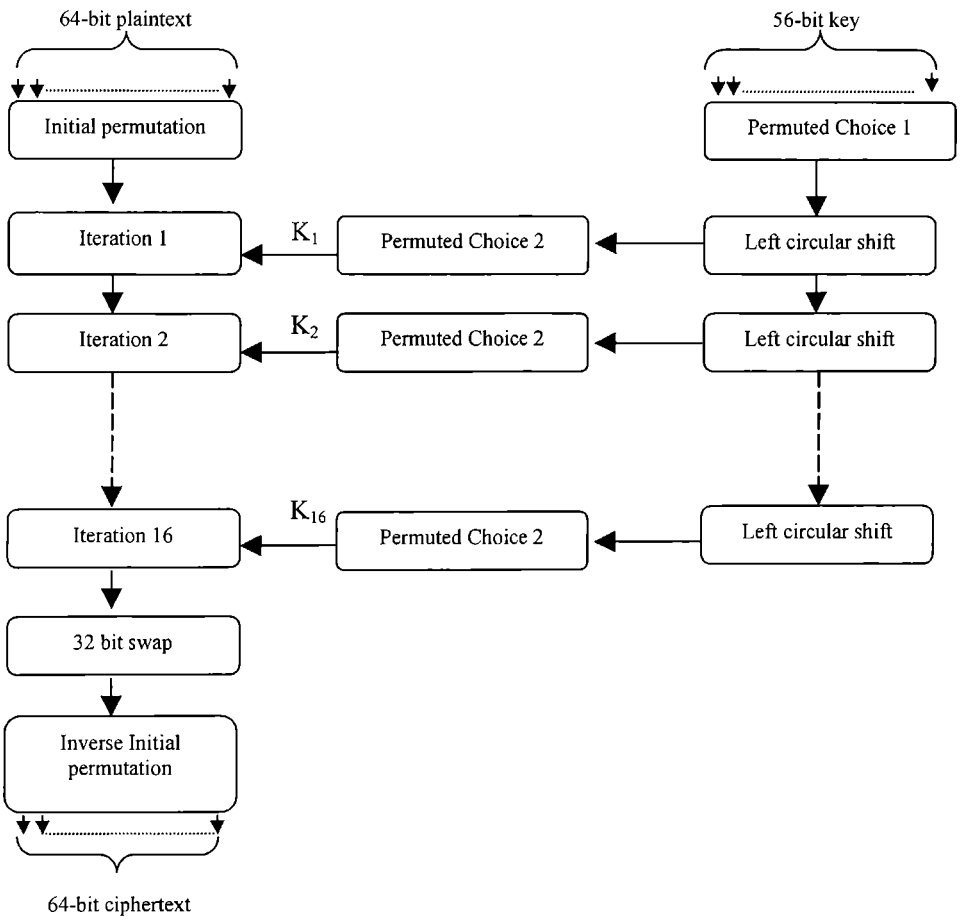
2.2.- DES (Data Encryption Standard)

DES es un mecanismo de encriptación de datos de uso generalizado. Hay muchas implantaciones de hardware y software de DES. Este transforma la información de texto plano en datos encriptados llamados texto cifrado mediante el uso de un algoritmo especial y de una clave. Si el receptor conoce la clave, podrá utilizarla para convertir el texto cifrado en los datos originales.

DES es el método de encriptación a utilizar en la versión 4 del Modelo de Autenticación y Autorización Kerberos , para encriptar tickets y claves.

Para el mecanismo de encriptación DES, los datos son encriptados en bloques de 64 bits usando clave de 56 bits. El algoritmo transforma 64 bits de entrada en una serie de pasos en una salida de 64 bits. Los mismos pasos, con la misma clave, son usados para la desencriptación.

2.2.1.- Encriptación DES



Esquema de Encriptacion DES

Hay dos entradas para la función de encriptación:

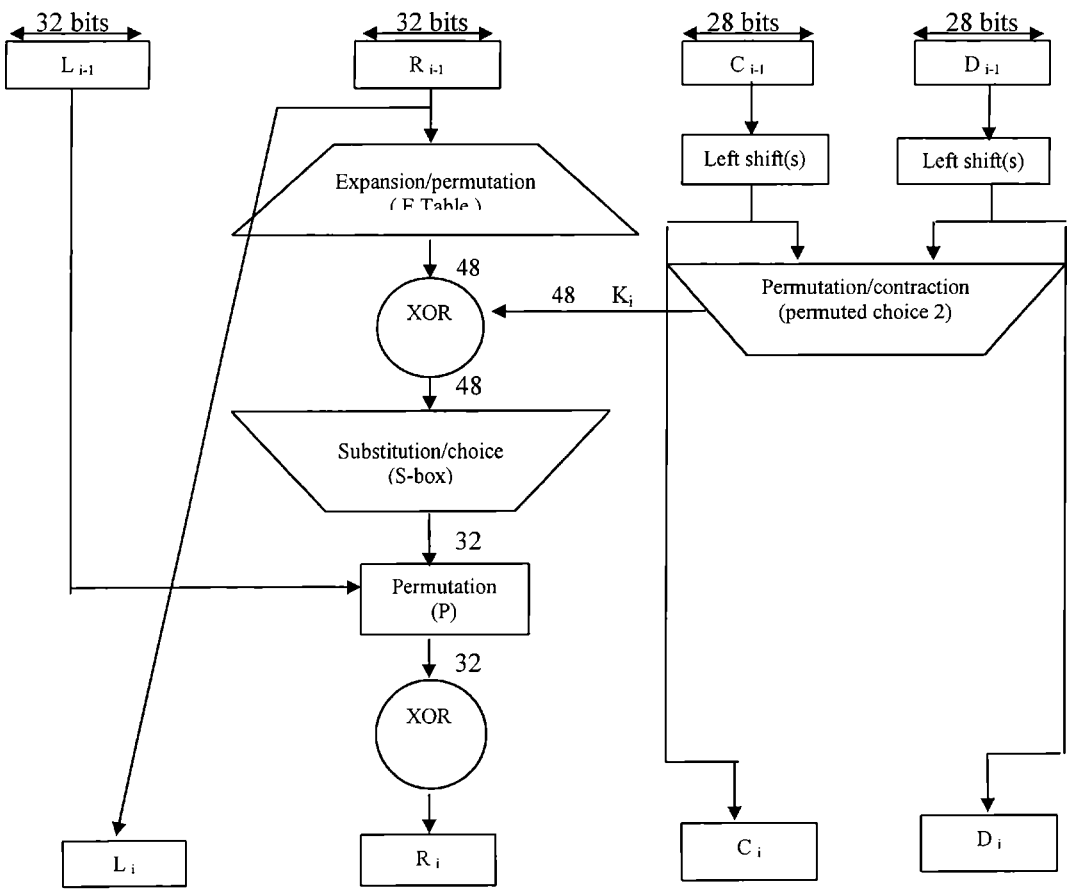
- el texto a ser encriptado (plaintext) de 64 bits de longitud, y
- la clave de 56 bits de longitud

Podemos ver el procedimiento del texto en tres fases:

1. El texto de 64 bits pasa a través de una permutación inicial (IP) que re-arregla la entrada permutada.
2. 16 iteraciones de una misma función que envuelve ambas: funciones de permutación y de sustitución. La salida de la iteración 16 consiste de 64 bits. La parte derecha y la parte izquierda de la salida son intercambiadas para producir el pre-output.
3. El pre-output es pasado a través de una permutación (IP^{-1}) que es la inversa de la función de permutación inicial que produce el texto de 64 bits encriptados.

¿Cómo se usan los 56 bits de la clave?

La clave es pasada a través de una función de permutación. Luego, para cada una de las 16 iteraciones, una subclase Ki es producida por la combinación de un corrimiento circular a izquierda y una combinación. La función de permutación es la misma para cada iteración, pero la subclave es diferente porque es producida por los corrimientos repetidos de los bits de la clave.



Simple Algoritmo de Iteración DES

2.2.1.1.- Detalles de una Simple Iteración

Los 64 bits de entrada permutados pasan a través de 16 iteraciones produciendo un valor intermedio de 64 bits como conclusión de cada iteración.

La parte izquierda y la parte derecha de cada valor intermedio de 64 bits se tratan en forma separada (32 bits cada uno), Left y Right.

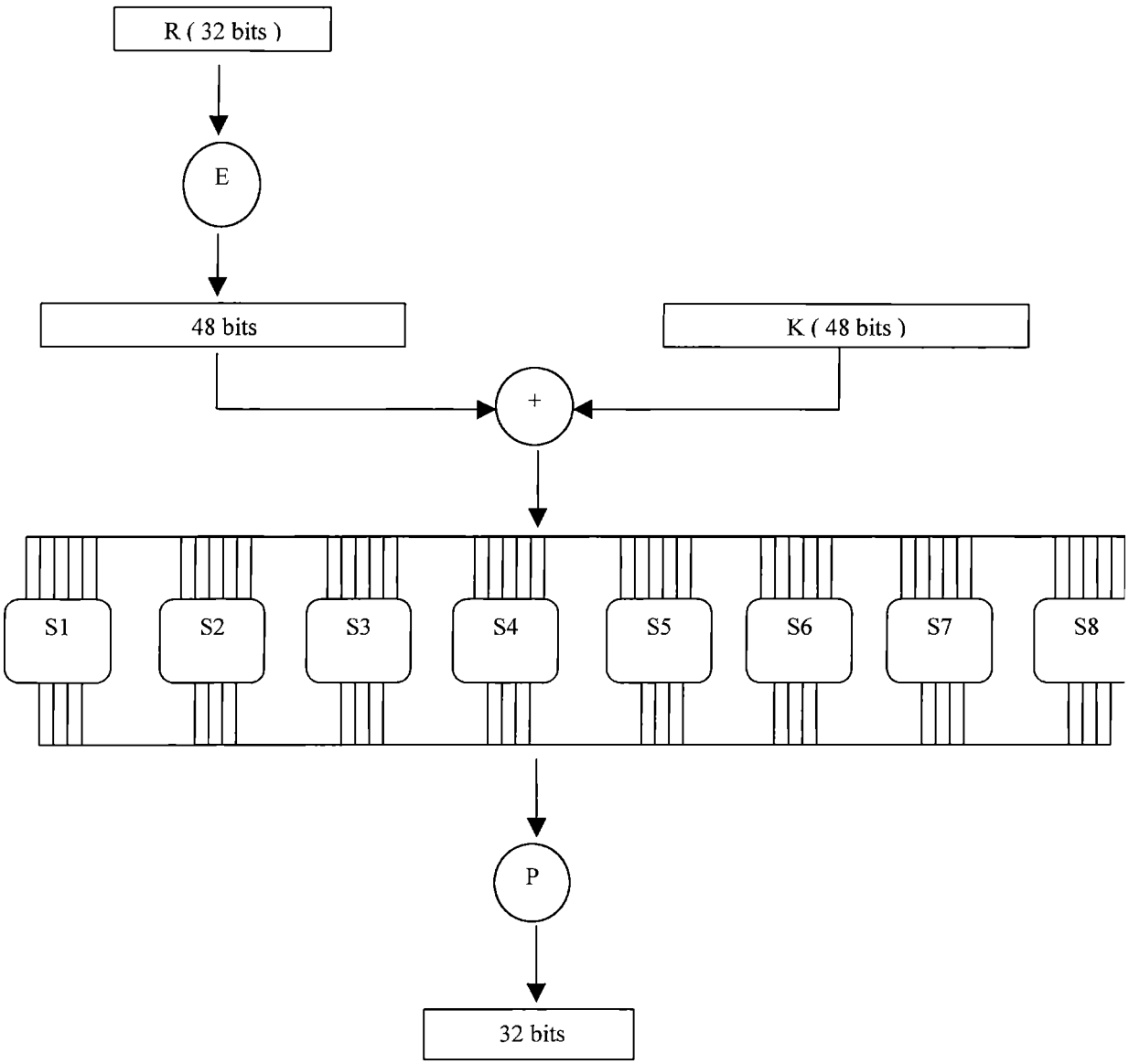
El procedimiento puede resumirse en las siguientes fórmulas:

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} @ f(R_{i-1}, R_i)$$

@ función XOR

La parte izquierda de la salida de una iteración (L_i) es simplemente igual a la parte derecha de la entrada de la iteración (R_{i-1})

La parte derecha de la salida es igual al or exclusivo de L_{i-1} y de la función f de R_{i-1} y K_i



Calculación de f (r,k)

La entrada R, de 32 bits, primero es expandida a 48 bits usando una tabla que define una permutación que expande a 16 bits mas de los bits R.

El resultado (XOR con la clave K_i), de 48 bits, pasa a través de una substitución que produce una salida de 32 bits, la cual es permutada

La función de substitución consiste de un conjunto de 8 boxes , cada uno de los cuales acepta 6 bits de entrada y produce 4 bits de salida.

El primer y último bit de salida del box S_i forma un número binario (2 bits) para seleccionar una fila particular de la tabla para S_i. Los 4 bits del medio seleccionan una columna en particular. El valor decimal en la celda seleccionada por una fila y una columna es luego convertida a sus 4 bits que representan la salida.

Por ejemplo: En S_1, para la entrada 011011, la fila 01 (fila 1) y la columna 1101 (columna 13), el valor de la fila 1 columna 13 es 5. Entonces la salida es 0101.

2.2.2.- Desencriptación DES

El proceso de desencriptación es esencialmente el mismo que el proceso de encriptación.

La regla es la siguiente:

Usar el texto encriptado como entrada al algoritmo DES, pero usar las claves K_i en orden inverso, o sea, usar la clave K_{16} sobre la primera iteración, K_{15} sobre la segunda, y así siguiendo hasta que K_1 sea usada sobre la 16 y última iteración. El mismo algoritmo con el orden de clave inverso produce el resultado correcto.

2.2.2.1.- Pasos del mecanismo de desencriptación

Tomar texto encriptado y usarlo como entrada al algoritmo DES.

Primer paso: pasar el texto a través del paso IP, produciendo 64 bits $L^d_0 \| R^d_0$ sabemos que IP es la inversa de IP^{-1}

Por lo tanto,

$$L^d_0 \| R^d_0 = IP(\text{texto encriptado})$$
$$\text{texto encriptado} = IP^{-1}(R^d_{16} \| L^d_{16})$$
$$L^d_0 \| R^d_0 = IP(IP^{-1}(R^d_{16} \| L^d_{16})) = R^d_{16} \| L^d_{16}$$

Entonces, la entrada del primer paso del proceso de desencriptación es igual a los 32 bits cambiados de la salida del paso 16 del proceso de encriptación.

Ahora podríamos mostrar que la S del primer paso del proceso de desencriptación es igual a 32 bits cambiados de la entrada de los 16 pasos del proceso de encriptación.

Primero consideramos proceso de encriptación :

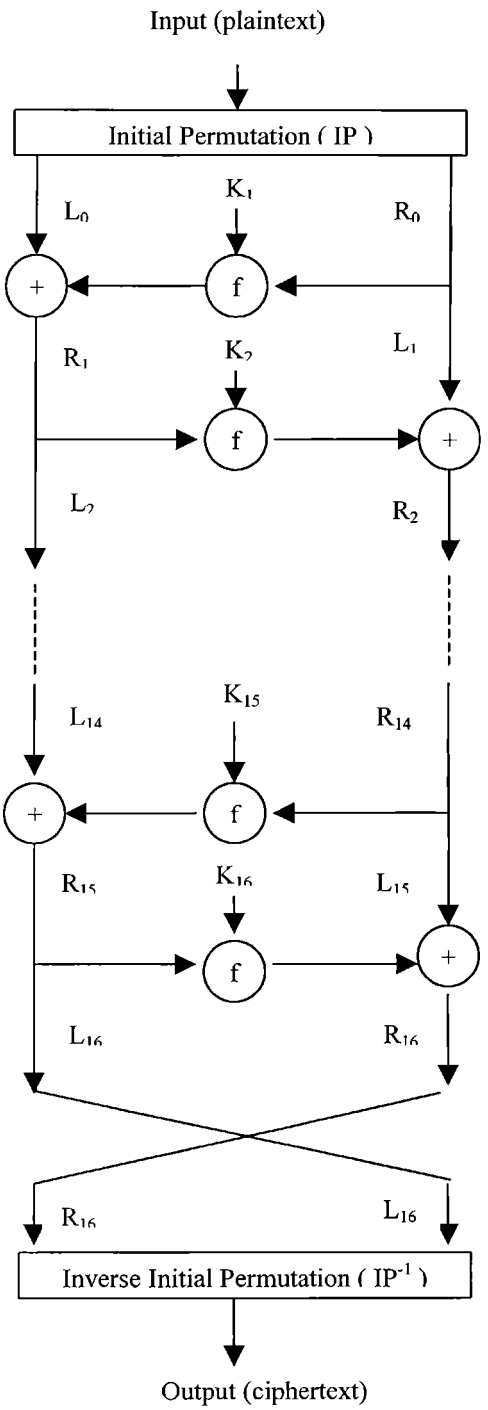
$$L_{16} = R_{15}$$
$$R_{16} = L_{15} @ f(R_{15}, R_{16})$$

sobre la desencriptación :

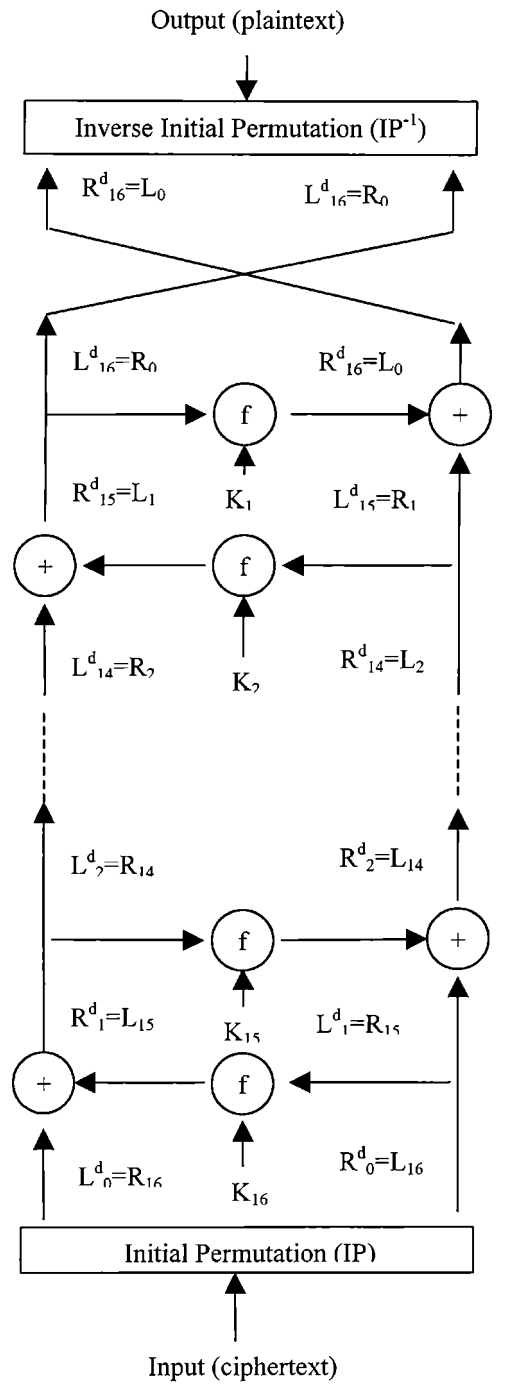
$$L^d_1 = R^d_0 = L_{16} = R_{15}$$
$$R^d_1 = L^d_0 @ f(R^d_0, K_{16})$$
$$= R_{16} @ f(R_{15}, K_{16})$$
$$= [L_{15} @ f(R_{15}, K_{16})] @ f(R_{15}, K_{16})$$

La siguiente figura muestra el proceso de encriptación del lado izquierdo y el proceso de desencriptación del lado derecho.

El diagrama indica que para todo paso el valor intermedio del proceso de desencriptación es igual al valor correspondiente del proceso de encriptación con las dos divisiones de valores cambiados. [1] [3] [22]



Encriptación DES



Desencriptación DES

2.3.- Otros mecanismos de Encriptación

2.3.1.- Algoritmo RSA

Otro algoritmo de encriptación es RSA, descubierto por el grupo MIT (1978), conocido por las iniciales de los 3 descubridores: Rivest, Shamir, Adleman (RSA).

Este método está basado en algunos principios de la teoría de números:

1) Elegir 2 números primos grandes , p y q (típicamente > 10^{100})
2) Computar n = p x q y z = (p-1) x (q-1)
3) Elegir un número relativamente primo a z y llamarlo d
4) Encontrar e tal que e x d = 1 mod z

Con estos parámetros podemos comenzar la encriptación. Divide el texto a encriptar en bloques, tal que cada texto p falle en un intervalo 0 <= p <= n, agrupando el texto en bloques de k bits, donde k es un entero grande por el cual $2^k < n$

Para encriptar un mensaje, P, computar $C = P^e \pmod n$
Para encriptar C, computar $P = C^d \pmod n$

Para todo el rango especificado, las funciones de encriptación y desencriptación son inversas.

Necesitamos e y n para la encriptación.
Necesitamos d y n para la desencriptación.
Por lo tanto, la clave pública consiste de un par (e, n)
Y la clave privada consiste de un par (d, n)

La seguridad de este método está basada sobre la dificultad del factoreo de números grandes. Si se podría factorear n (conocida públicamente) podría encontrar luego p y q, y desde ellos z. Equipados con z y e, d puede ser encontrado usando algoritmo Euclides. Afortunadamente, los matemáticos han probado factorear números largos en un mínimo de 300 años, y es excesivamente dificultoso.

De acuerdo con Rivest y colegas, factorear un número de 200 dígitos requiere 4 billones de años de tiempo computado. Factorear un número de 500 dígitos requiere 10^{25} años. En ambos casos, ellos asumen el mejor algoritmo conocido. [3]

2.3.2.- Algoritmo IDEA

IDEA International Data Encryption Algorithm (1990 – 1992)

IDEA (International Data Encryption Algorithm) fue diseñado en Switzerland (1990-1992).

Es un algoritmo de encriptación convencional orientado a bloques,desarrollado por Xuejia Lai y James Massey de Swiss Federal Institute of Technology. Usa claves de 128 bits.

IDEA es uno de los algoritmos de encriptación convencional que ha sido propuesto en años recientes para reemplazar DES.

IDEA encripta bloques usando claves de 128 bits para encriptar datos en bloques de 64 bits. En contraste, DES también usa bloques de 64 bits, pero con clave de 56 bits.

Hay dos entradas para la función de encriptación:

- el texto a ser encriptado, de 64 bits de longitud, y
- la clave, de 128 bits

El algoritmo consiste de 8 iteraciones seguidos por una función de transformación. El algoritmo divide la entrada en sub-bloques de 16 bits. Cada uno de las iteraciones toma 4 sub-bloques de 16 bits como entrada y produce 4 bloques de 16 bits como salida.

La transformación final también produce bloques de 16 bits los cuales son concatenados para formar el texto encriptado de 64 bits.

Cada uno de las iteraciones hace uso de 6 sub-claves de 16 bits donde la transformación final usa 4 sub-claves, para un total de 52 sub-claves.

Las 52 sub-claves son generados por la clave original de 128 bits. [3]

Capítulo 3

Protocolo de Autenticación y Autorización Kerberos

3.1.- Introducción

Internet es un lugar inseguro. Muchos de los protocolos usados en Internet no proveen ningún tipo de seguridad. Los sistemas *crackers* usan comúnmente las herramientas para descubrir *passwords*. De esta manera las aplicaciones que envían una password no encriptada sobre la red son extremadamente vulnerables. En el peor de los casos, otras aplicaciones cliente/servidor confían en que el programa cliente sea honesto con respecto a la identidad del usuario que lo esta utilizando. Otras aplicaciones confían en que el cliente restrinja sus actividades a aquellas que tenga permitido hacer, sin forzar esto en el servidor.

En Internet, una estación de trabajo no puede ser confiable para identificar a sus usuarios correctamente para servicios de red. En particular puede suceder lo siguiente:

- Un usuario puede ganar el acceso a una estación de trabajo en particular y simular ser otro usuario operando desde otra estación de trabajo.
- Un usuario puede alterar la dirección de red de una estación de trabajo para que los requerimientos que envía simulen llegar desde otra estación de trabajo
- Un usuario puede *'escuchar disimuladamente'* los intercambios y atacar para ganar la entrada a un servidor o para interrumpir operaciones.

En muchos de estos casos, un usuario podría ganar acceso a servicios y datos que no esta autorizado a acceder.

Algunos sitios intentan usar *firewalls* para resolver los problemas de seguridad en sus redes. Desafortunadamente los *firewalls* asumen que los problemas están del lado de afuera, lo cual no siempre es cierto. Además, estos tienen una desventaja significativa: restringen la forma en la que los usuarios pueden utilizar Internet. En muchos de los casos utilizan suposiciones que son irrealistas e improbables.

Kerberos nace como una solución al problema de seguridad de red, fue creado por MIT (Massachusetts Institute of Technology) - Proyecto ATHENA. Este protocolo usa fuertemente la criptografía y permite que un cliente pueda demostrar su identidad a un servidor y viceversa, a través de una conexión de red insegura. Las claves proveen la base para la autenticación en Kerberos.

Kerberos usa encriptación simétrica, método en el cual la encriptación y desencriptación se realizan usando una única clave.

Después que un cliente y un servidor han usado Kerberos para verificar su identidad, ellos pueden encriptar todas sus comunicaciones para asegurar integridad y privacidad en los datos.

El origen del nombre Kerberos proviene de la mitología griega y *significa "un perro de muchas cabezas, comunmente tres, quizás con una cola de serpiente, el guardián de la entrada de Hades".*

Así, como el Kerberos de Grecia tiene tres cabezas, el Kerberos moderno fue pensado para tener tres componentes para cuidar una puerta de red:

- Servidor de Autenticación (AS),
- Servidor de Concesión de Tickets (TGS), y
- Servidor de Auditoría (KADM)

Kerberos es una solución a los problemas de seguridad en la red. Provee las herramientas de autenticación y fuerte criptografía sobre la red para ayudar a mantener segura la información que circula.

Es un protocolo de autenticación y autorización para redes de computadoras que esta basado en el protocolo de claves de distribución de Needham y Schroeder.

Autenticación: es la verificación de la identidad de la parte que ha generado los datos y la integridad de los mismos, su propósito básico es impedir pedidos de conexiones fraudulentos.

Autorización: es el proceso por el cual se determina de algún modo si un principal puede realizar una operación.

Usualmente la autorización se hace antes que la autenticación.

En lugar de construir protocolos de autenticación en cada servidor, Kerberos provee un servidor de autenticación centralizado, cuya función es autenticar usuarios frente a servidores y servidores frente a usuarios.

Kerberos es ampliamente usado hoy en día como un buen sistema de seguridad de Internet por instituciones financieras, agencias gubernamentales, y universidades. También es muy usado en el protocolo de nivel de aplicación (modelo ISO nivel 7) tal como TELNET o FTP para proveer seguridad. Es también usado en forma menos frecuente como sistema de autenticación implícito de data stream, tal como SOCK-STREAM o mecanismos RPC (modelo ISO nivel 6). También puede usarse en protocolos IP,UDP,TCP. [1] [9] [10] [3]

3.2.- Motivación

En un ambiente de computadoras personales sin red ,los recursos e información pueden ser protegidas asegurando físicamente cada computadora personal. En un ambiente de tiempo compartido , los sistemas operativos protegen a los usuarios y a los recursos desde otra computadora. Para determinar qué es lo que cada usuario esta habilitado para leer o modificar , es necesario que el sistema de tiempo compartido identifique cada usuario. Esto se logra cuando el usuario se conecta (login).

En una red de usuarios que requieren servicios desde muchas computadoras separadas, hay tres enfoques donde uno puede tomar el control del acceso: uno no puede hacer nada, confiando en la máquina a la cual el usuario esta conectado para prevenir el acceso no autorizado; otro puede hacer que el host pruebe su identidad , pero solo confía en la palabra del host en cuanto a quien es el usuario; otro puede hacer que el usuario pruebe su identidad para cada servicio requerido.

En un ambiente cerrado donde todas las máquinas estan bajo estricto control, se puede usar el primer enfoque . Cuando la organización controla todos los hosts comunicados sobre la red, esto es un enfoque razonable.

En un ambiente mas abierto, uno puede selectivamente confiar solo en aquellos host bajo control organizativo. En este caso cada host debe probar su identidad. La autenticación es realizada chequeando las direcciones de Internet desde la cual la conexión ha sido establecida.

En un ambiente Athenas debemos habilitar los pedidos desde el host. Los usuarios tienen control completo de sus estación de trabajo; ellos pueden entre otras cosas reencenderla, hacer que funcionen solas. De esta manera se debe tomar el tercer enfoque. El usuario debe probar su identidad para cada servicio deseado. Esto no es suficiente para asegurar fisicamente que el host corra en un server de la red. Alguien en otro lado de la red puede estar camuflándose como el servidor dado.

Nuestro ambiente establece varios requerimientos sobre un mecanismo de identificación:

- Primero, debe ser seguro. Debe ser lo suficientemente dificil de engañar como para que un hacker potencial no encuentre que el mecanismo de autenticación es el enlace débil. Alguien que observa la red no debería ser capaz de obtener la informacion necesaria para impersonalizar otro usuario.
- Segundo, debe ser confiable. El acceso a muchos servicios dependerá del servicio de autenticación, si esto no es confiable el sistema de servicios en su conjunto no lo será.
- En tercer lugar, debe ser transparente. Idealmente el usuario no debería darse cuenta de que la autenticación está sucediendo.
- Finalmente, debería ser escalable. Muchos sistemas pueden comunicarse con los host de Athenas, no todos ellos pueden soportar nuestro mecanismo, pero el software no debería romperse si lo soportara.

Kerberos es el resultado de nuestro trabajo para satisfacer los requerimientos mencionados con anterioridad. Cuando un usuario se acerca a una estación de trabajo se conecta (login). En cuanto a lo que el usuario pueda decir, la identificación inicial es suficiente para probar su identidad a todos los servicios de red requeridos para la duración de la sesión de conexión. La seguridad de Kerberos confia en la seguridad de varios servidores de autenticación, pero no en el sistema desde el cual los usuarios se conectan, tampoco en la seguridad de los servidores finales que serán usados.

La autenticación es un bloque de construcción esencial para un ambiente de red seguro. Si por ejemplo, un server conoce con certeza la identidad de un cliente puede decidir si brindar el servicio o si el usuario deberia recibir privilegios especiales, quien debería recibir un ticket por el servicio, y así sucesivamente. En otras palabras los esquemas de autorización pueden ser construidos sobre la autenticación que Kerberos provee, dando como resultado una seguridad equivalente a una computadora personal sola o sistema de tiempo compartido.

3.3.- Modelo de Autenticación Kerberos

Kerberos trabaja proveyendo a los usuarios con tickets y claves encriptadas llamadas *claves de sesión* que pueden usar para identificarse con otros usuarios o servicios en una red.

Hay tres fases para autenticar a través de Kerberos:

- En la primer fase, el usuario obtiene credenciales a ser usadas para pedir accesos a otros servicios.
- En la segunda fase, el usuario pide autenticación para un servicio específico.
- En la fase final, el usuario presenta estas credenciales para terminar con el servicio.

Hay dos tipos de credenciales usadas en el modelo de autenticación Kerberos: *Tickets* y *Autenticadores,* ambos basados en la encriptación DES que usa la misma clave para encriptar que para desencriptar.

Un ticket es usado para asegurar el pase de la identidad de la persona que emitió el ticket entre el server de autenticación y el server final, también nos garantiza que la persona que está usando el ticket es la misma que lo emitió.

La información que está impresa en el ticket es: el nombre del server, el nombre del cliente, la dirección de internet del cliente, un timestamp, un tiempo de vida, y una clave de sesión aleatoria. Esta información es encriptada usando la clave del servidor. El ticket emitido puede ser usado varias veces por el cliente nombrado para acceder al server dado, hasta que el mismo expire. Notar que como el ticket es encriptado con la clave del server , permite al usuario pasar el ticket sobre el server sin tener que preocuparse que el usuario modifique el ticket, la información que el ticket contiene:

{ server, cliente, dirección de red de cliente, timestamp,tiempo de vida, clave privada entre el server y el cliente }, todo encriptado con la clave privada del server (K_S)

Un nuevo ticket debe ser generado cada vez que el cliente quiere usar el servicio.

La responsabilidad de imprimir los tickets no es de la computadora local sino del Server Kerberos referenciado como 'Key Distribution Center'. Esta maquina es la más segura de la red y almacena la clave de los usuarios.

El autenticador se utiliza para poder verificar que el cliente es quien dice ser y contiene información adicional, dicha información es la siguiente:

{ nombre del cliente, la dirección IP de la estación de trabajo, y el tiempo corriente de la estación de trabajo } , encriptado con la clave de sesión.

Un enfoque básico para la autenticación Kerberos es el siguiente: un cliente para poder usar un servicio determinado debe tener un ticket previamente obtenido desde un Servidor Kerberos. Un ticket para un servicio es un string que ha sido encriptado usando la clave privada para el servicio. El servicio puede confiar que la información dentro del ticket fue originada por Kerberos porque la clave privada solo es conocida por el servicio mismo y Kerberos.

El cliente obtiene un ticket enviando un mensaje a Kerberos, dicho mensaje contiene: el identificador del servicio deseado, el identificador del cliente y un tiempo corriente.

Kerberos envía una respuesta sólo usable por el cliente que originó el pedido ya que la misma está encriptada con la clave privada del cliente. La respuesta contiene tres partes: el ticket, la clave de sesión y un timestamp impreso por el server Kerberos.

Un usuario legítimo puede desencriptar el mensaje para obtener el ticket y la clave de sesión y poder verificar que el timestamp sea corriente.

El cliente envía el ticket acompañado del autenticador al Server de Servicio.

El autenticador mencionado consiste: el identificador del cliente principal, la direccion de network y un tiempo corriente todo encriptado con la clave de sesión.

Luego el Server de Servicio desencripta el ticket, usando la clave de sesion desencripta el autenticador y verifica si el identificador del cliente en el ticket es igual al del autenticador, la dirección de network que le envió el mensaje es igual a la que está en el autenticador y el tiempo en el autenticador corresponde, entonces el autenticador probablemente no sea una copia y el servicio acepta el ticket asociado.

Si un mismo usuario desea utilizar más de un servicio, necesitará un ticket para cada servicio.

3.3.1.- Un Simple Modelo de Autorización

En una ambiente de red no protegida, cualquier cliente puede solicitar servicio a cualquier server. Un *hacker* puede aparentar ser otro cliente y obtener privilegios no autorizados sobre el server.

Los servers deben estar disponibles para confirmar la identidad de los clientes que piden servicios.

Una alternativa es usar un server de autenticación (AS) que conozca la password de todos los usuarios y almacenarlos en una base de datos centralizada.

El AS comparte una clave secreta única con cada server. Estas claves han sido distribuídas físicamente o de alguna manera en forma segura.

Ahora consideramos el siguiente diálogo:

1) C → AS ID_C, P_C, ID_V
2) AS → C Ticket
3) C → V ID_C , Ticket

Ticket = $E_{Kv}[ID_C , AD_C, ID_V]$

C : Cliente
AS : Server de Autenticación
V : Server

El usuario se conecta en la estación de trabajo y pide acceso al server V.

El cliente en la estación de trabajo pide la password del usuario y luego envía el mensaje el mensaje al AS que incluye el ID_C , el ID_V y la password del usuario.

El AS chequea su Base de Datos. Si ambos testeos son pasados, el AS acepta al usuario como auténtico y ahora debe convencer al server que este usuario es auténtico. El AS crea un ticket que contiene el ID del usuario, la dirección de red (network address) y el ID del Server. El Ticket es encriptado usando la clave secreta compartida por el AS y el Server. Este ticket es luego enviado a C. Como el ticket es encriptado no puede ser alterado por C o por un *hacker*. Con este ticket, C puede ahora pedir al Server el servicio. C envía un mensaje al Server conteniendo el ID del cliente y el ticket. El

Server desencripta el ticket y verifica que el usuario ID en el ticket sea el mismo que el user ID desencriptado en el mensaje. Si estos dos coinciden, el Server considera al mismo autenticado y otorga el servicio pedido.

El ticket es encriptado para prevenir alteración o falsificación. El ID_V es incluido en el ticket. El ID_C está incluido en el ticket para indicar que este ticket ha sido impreso en nombre de C.

Finalmente, AD_C (dirección de red de C) sirve para salvar las siguientes amenazas. Un hackers podría capturar el ticket transmitido en el mensaje 2, luego usa el nombre ID_C y transamite un mensaje de forma 3 desde otra estación de trabajo. El Server recibiría un ticket válido que coincide con el ID_C y otorga acceso al usuario sobre otra estación de trabajo. Para prevenir este ataque, el AS incluye en el ticket la dirección de red desde el cual se originó el pedido. Ahora el ticket es válido solo si este es transmitido desde la misma estación de trabajo que inicialmente pidió el ticket.

Primer problema: queremos minimizar la cantidad de veces que el usuario entra la password. Suponemos que cada ticket puede ser usado solo una vez. Si el usuario C se conecta a la estación de trabajo a la mañana y desea chequear su mail en el mail server , C debe suministrar la password tomando un ticket por el mail server. Si desea chequearse varias veces al día, cada intento requiere re-entrar la password.

Para una simple sesión de logon, la estación de trabajo puede almacenar el ticket del mail server despues que haya sido recibido y usarlo en nombre del usuario para múltiple accesos al mail server.

Si embargo, bajo este esquema recordamos el caso que el usuario necesitara un nuevo ticket por cada diferente servicio. Si el usuario desea acceder al print server, al mail server, al file server, y demás, la primera instancia de cada acceso requiere un nuevo ticket y así requiere al usuario entrar la password.

Segundo problema: transmisión de la password en texto plano (mensaje 1). Alguien podría capturar la password y usarla para cualquier servicio accesible a la víctima.

3.3.2.- Un Modelo más seguro de Autenticación

Técnica de Distribución de la Clave de Sesión.

El Cliente C envía un mensaje al AS pidiendo acceso al TGS.

AS responde a C con un mensaje encriptado con la clave derivada de la password de usuario (K_C) , que contiene el ticket.

El mensaje encriptado también contiene una copia de la clave de sesión ($K_{C,TGS}$) (esta clave de sesion es para C y TGS).

Como la clave de sesión se encuentra dentro del mensaje encriptado con la clave del cliente K_C, solo el usuario cliente puede leerlo. La misma clave de sesión es incluida en el ticket, el cual puede ser leída solo por TGS.

La clave de sesión ha sido seguramente entregada a ambos C y TGS. El mensaje 1 incluye un timestamp (TS_1). En el mensaje 2 el ticket incluye varios elementos. Esto habilita a C a confirmar que este ticket es para TGS y toma su tiempo de expiración. Ahora armado con el ticket y la clave de sesión , C está listo para aproximarse a TGS. C envía a TGS un mensaje que incluye el ticket más el ID del servicio pedido (mensaje 3)

C transmite un autenticador que incluye el ID y dirección del usuario cliente y un timestamp.

TGS puede desencriptar el ticket con la clave que éste almacena con el AS. El ticket indica que el usuario C ha sido provisto con la clave de sesión $K_{C,TGS}$ (cualquiera que use $K_{C,TGS}$ debe ser C).

TGS usa la clave de sesión para desencriptar el autenticador. TGS puede luego chequear el nombre y dirección desde el mensaje entrante. Si todos coinciden, luego el TGS está seguro que el cliente que envió el ticket es realmente el que lo envió.

La respuesta desde el TGS (mensaje 4) sigue la forma del mensaje 2. El mensaje es encriptado con la clave del mensaje compartido por TGS y C, incluye una clave de sesión compartida entre C y V, el ID de V y el timestamp del ticket.

El ticket mismo incluye la misma clave de sesión.

C ahora tiene un TGS reusable para V.

Cuando C presenta este ticket (mensaje 5), éste también envía un autenticador. El Server puede desencriptar el ticket, recupera la clave de sesión y desencripta el autenticador.

Si la autenticación mutua es requerida, el Server puede responder (mensaje 6).

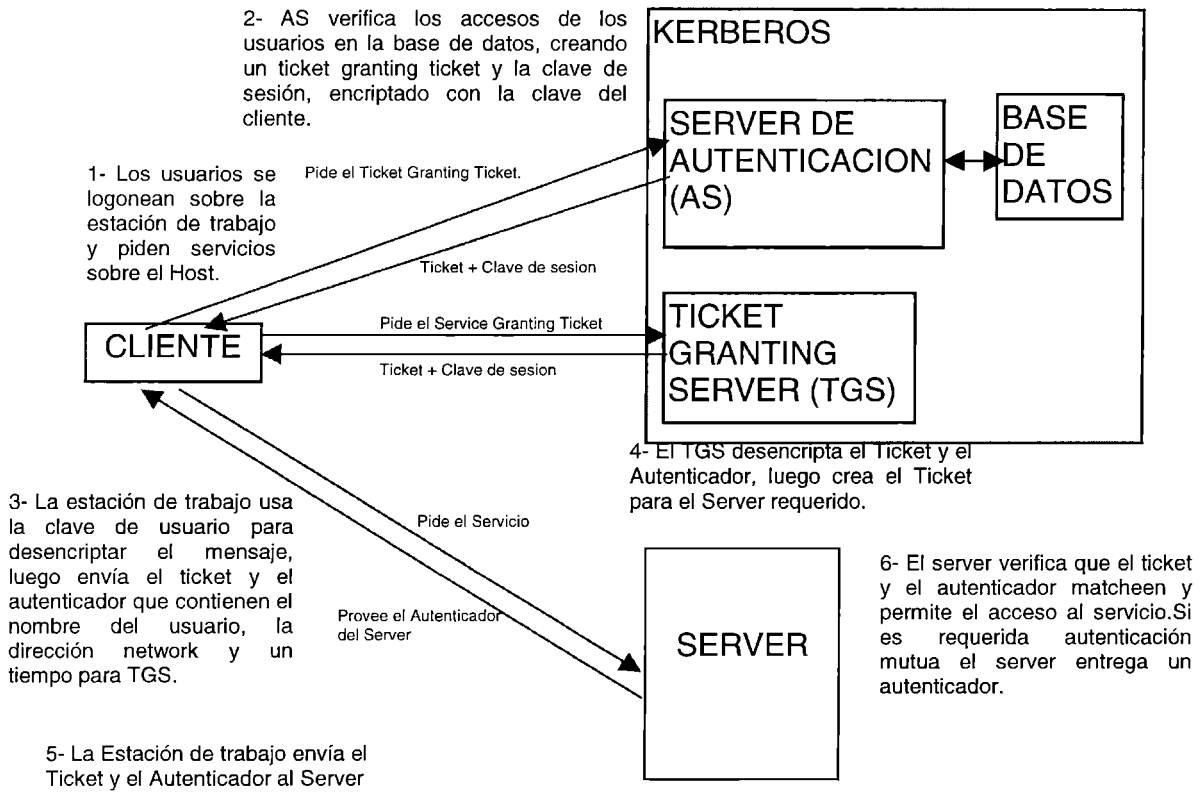
El Server retorna el valor del timestamp desde el autenticador, incrementado en 1 y encriptado con la clave de sesión.C asegura que éste podría haber sido creado solo por V. El contenido del mensaje asegura a C que esta no es una repuesta vieja.

Finalmente, el Cliente y el Server comparten una clave secreta. Esta clave puede ser usada para encriptar mensajes futuros entre los dos o intercambiar una nueva clave de sesión random para éste propósito.

Para este problema introducimos un esquema evitando password de texto plano y un nuevo server, conocido como ticket-granting server (TGS).

TGS imprime tickets a usuarios que han sido autenticados al AS. Así, el usuario primero pide ticket-granting ticket ($Ticket_{TGS}$) desde el AS. Este ticket es salvado por el módulo cliente en la estación de trabajo del usuario. Cada vez que el usuario requiera acceso al nuevo servicio, el cliente suministra al TGS usando el ticket para autenticarse. El TGS luego otorga un ticket para un particular servicio. El cliente salva cada ticket service-granting y usa éste para autenticar su usuario al server cada vez que un servicio es requerido. Un problema sería el tiempo de vida asociado con el ticket-granting ticket. Si es un tiempo corto (minutos) el usuario debería pedir repetidamente la password. Si es largo (horas) un *hacker* tiene gran oportunidad para responder, por lo general el tiempo que se le otorga al ticket granting ticket es de 8 horas.

3.3.3.- Esquema de la Version 4 del Modelo de Autenticacion Kerberos (Intercambio de Mensajes)



a) Servicio de Autenticación para obtener un ticket-granting ticket

(1) C → AS : IDc I IDtgs I TS1
(2) AS → C : Ekc [K c,tgs I IDtgs I TS2 I tiempo de vida I Ticket tgs I
Ticket tgs : Ektgs[K c,tgs I IDc I ADc I IDtgs I TS2 I tiempo de vida2I

b) Ticket Granting Service para obtener el servicet granting ticket
(3) C → TGS : IDv I Ticket tgs I Autenticador c
(4) TGS → C : Ekc,tgs [Kc,v I IDv I TS4 I Ticket v]
Ticket tgs : Ektgs [Kc,tgs I IDc I ADc I IDtgs I TS2 I tiempo de vida2]
Ticket v : Ekv [Kc,v I IDc I ADc I IDv I TS4 I tiempo de vida4]
Autenticador c : Ekc,tgs [IDc I ADc I TS3]

c) Cliente/Server, intercambio para obtener el servicio

(5) C → Server : Ticket v I Autenticador c
(6) Server → C : Ekc,v [TS5 + 1I
Ticket v : Ekv [Kc,v I Idc I Adc I Idv I TS4 I tiempo de vida4 I
Autenticador c : Ekc,v [IDc I ADc I TS5 I

3.3.4.- Ejemplo de cómo un cliente pide un servicio a Kerberos

Kerberos envuelve tres Servidores:

- *Server de Autenticación (**AS**):* verifica a los usuarios durante la conexión
- *Server Ticket-Granting (**TGS**)* : emite 'pruebas de tickets de identidad'
- *Bob, el Server*

Alice es una estación de trabajo cliente.

AS es similar a un KDC (Key Distribution Center) en el que comparte una password secreta con todos los usuarios. El trabajo de TGS es emitir tickets que pueden convencer a los servers real que el portador de los tickets TGS realmente es quien dice ser.

Para comenzar una sesión, Alice se detiene en una estación de trabajo pública arbitraria y tipea su nombre. La estación de trabajo envía su nombre al AS en texto plano. Vuelve una clave de sesión y un ticket K_{TGS} (A,K_S) entendido por el TGS . Estos items son empaquetados juntos y encriptados usando la clave secreta de Alice, que solo Alice puede desencriptarla.

Solo cuando el mensaje 2 llega, la estación de trabajo pide la password de Alice, usada luego para generar K_A , para desencriptar el mensaje 2 y obtener la clave de sesión y la parte interna del ticket TGS .

A este punto, la estación de trabajo sobreescribe la password de Alice , asegura que está solo dentro de la estación de trabajo por pocos milisegundos. Si Trudy intenta logonearse como Alice , la password que ella tipea debe ser errónea y la estación de trabajo debería detectar esto porque la parte standard del mensaje 2 debería ser incorrecta.

Después que ella se conecte, Alice puede decirle a la estación de trabajo que quiere contactarse al Server de Bob. La estación de trabajo luego envía el mensaje 3 a TGS preguntando por el ticket a usar con Bob. El elemento clave en este pedido es K_{TGS} (A,K_S), el cual está encriptado con la clave secreta de TGS, y usado como prueba que el enviador es realmente Alice.

TGS responde creando una clave de sesión K_{AB} para que Alice la use con Bob. Dos versiones son enviadas de retorno: la primera encriptada con solo K_S para que Alice pueda leerlo. La segunda encriptada con la clave de Bob, K_B, para que Bob pueda leerlo. Trudy puede copiar el mensaje 3 y usarla, pero debería ser frustrado por el timestamp encriptado t, solo enviado con éste. Trudy no puede reemplazar el timestamp por uno más reciente, porque no conoce K_S , la clave de sesión que Alice usa para hablar a TGS. Si Trudy replica el mensaje 3 rápidamente, debería tomar otra copia del mensaje 4, el cual ella no podría desencriptar la primera vez y no debería estar disponible para desencriptar la segunda vez.

Ahora Alice puede enviar K_{AB} a Bob para establecer una sesión con él. Este intercambio también tiene un timestamp. La respuesta es prueba que Alice está actualmente comunicado con Bob, no con Trudy. Después de estas series de intercambios , Alice puede comunicarse con Bob a cubierto de K_{AB} . Si ella más tarde decide que necesita comunicarse con otro Server, Carol, ella repite exactamente el mensaje 3 a TGS, ahora solo especificando C instanciado en B. TGS prontamente

debería responder con un ticket encriptado con K_C que Alice puede enviar a Carol y que Carol debería aceptar como prueba que este vuelve desde Alice. [3] [8] [9] [10] [11] [21]

3.4.- Modelo de Autorización Kerberos

El modelo de autenticación Kerberos provee sólo una certificación de la identidad de los pedidos de un cliente, pero no provee información a cerca de si el cliente está o no actualmente autorizado para usar el servicio. Hay tres formas en el cual la autorización podría ser integrada con el modelo de autenticación Kerberos:

- La Base de Datos Kerberos podría también contener información de autorización para cada servicio y emitir tickets de servicios solo a usuarios autorizados de cada servicio.
- Un servicio de autorización separado podría mantener información de autorización teniendo acceso a las listas de cada servicio y permitiendo al cliente obtener certificación sellada de los miembros de la lista.. El cliente presentaría esta certificación al último servicio, en vez de un ticket Kerberos.
- Cada servicio podría mantener su propia información de autorización con la ayuda auxiliar de un servicio que almacena listas públicas compartidas y proveen certificación de la lista publica.

La primera de estas alternativas establece la gran Base de Datos de autorización dinámicamente actualizada. Los parámetros operativos tal como el tamaño de la memoria primaria y secundaria, el grado de replicación, la naturaleza del *backup*, y la seguridad fisica debe ser elegido como un compromiso entre los requerimientos de los dos servicios. También se encierra en un modelo de autorización particular para todas las aplicaciones.

La segunda alternativa separa la base de datos de la autorización de la Base de Datos de autenticación, por lo tanto mejorando la separación de la administración y haciendo el servicio de autenticación más simple y pequeño, el cual debería hacerlo más confiable y más fácil de asegurar. Pero esta alternativa conduce a una extraordinariamente compleja (por lo tanto, potencialmente frágil) colección de protocolos interactuantes entre el cliente y la autenticación, autorización y servicios destinos. También crea un problema rendezvous, en el cual el cliente debe conocer cuál es la certificación de membresía para pedir desde el server de autorización.

El modelo de autorización Kerberos está basado en el principio que cada servicio conoce mejor quiénes deberían ser sus usuarios y qué forma de autorización es la apropiada, para que adopte la tercera de estas alternativas .

Es importante notar que actualmente Kerberos no provee la autorización misma, pero permite a otras aplicaciones la habilidad para realizar una autorización.

Esta elección tiene varias ventajas :

- Muchos servicios tendrán listas cortas y privadas de usuarios autorizados. Por ejemplo, el *server display* en una estación de trabajo privada puede tener como lista de usuarios autorizados solo una entrada (el usuario actual de la

estación de trabajo) y que la identidad del usuario sea ya conocida por la estación de trabajo. La manera más simple de manejar esa información es situarla en el servidor. Completamente los servicios privados requieren una registración no central, aún puede tomar ventajas de la autenticación de la calidad Kerberos e implementar el control de acceso.

- Los servicios que mantienen su propia lista (por ejemplo, el servidor display) o que no requieran un acceso a la lista de control (por ejemplo, una biblioteca pública) no depende de la disponibilidad y continuidad de la red a un servicio de autorización.
- El problema *Rendezvous* es limitado para obtener al cliente junto con el servicio, el cliente no necesita imaginarse qué clase de autorización pide para este servicio particular.
- Ningún modelo de autorización se aplica a todos los servicios; autorizando la responsabilidad del server , el diseñador del servicio tiene la opción de usar el modelo de autorización estándar de las bibliotecas o crear un modelo diferente que se adapte mejor al servicio en particular que ofrece.
- Ya que la cantidad de almacenamiento de la información requerida para la información de autorización es proporcional al número de servicios ofrecidos, almacenando y administrando la información de autorización en el servicio se amplía bien. La ventaja de esta ampliación es de interés particular cuando uno realiza que toda estación de trabajo exporta al menos su servicio display, y puede exportar otros. Es también administrativamente preferible hacer que cada servicio provea su propio almacenamiento de lista de autorización, en vez de cargar el almacenamiento público con esta responsabilidad.
- La autoridad administrativa para establecer y cambiar la información de autorización para el servicio tiende a ser automáticamente delegada a una entidad apropiada (la administración del servicio mismo)

Hay una desventaja significativa al requerir que el servicio realice su propia autorización: los servicios que no pueden depender de otros servicios de red (por ejemplo, porque son un *single-threaded* y no debería bloquear esperando una repuesta de red) no puede hacer uso de las listas de control de acceso público compartido.

3.4.1.- Mecánicas de Autorización

Se provee un modelo de autorización standard basado en la lista de control de acceso, y esta disponible un paquete de biblioteca de autorización para la incorporación en cualquier servicio que encuentre útil el modelo standard. Bajo este modelo, el servicio toma la identidad del cliente y verifica que el cliente sea o no miembro de la lista nombrada. El paquete de librerías de listas de acceso mantiene cualquier número de listas nombradas en el almacenamiento local del server. Una lista puede contener tres clases de nombre:

- Identificadores de los *principals* de Kerberos autenticables,
- Nombres de otras listas locales,
- Nombres de listas de control de acceso público compartidas

El acceso a la lista de librerías asume una búsqueda de la lista nombrada, sublistas locales almacenadas en el host del servicio, y listas públicas compartidas. Si la identidad del cliente es encontrada en la búsqueda, la operación es autorizada.

En vez de asociar permisos de operación específica con entradas a la lista de acceso, el servicio mantiene distintas listas de acceso nombradas para cada tipo de operación diferente.

Las listas se mantienen como un simple archivo de string de texto ASCII en un directorio de lista de acceso especial que esta protegido de la modificación excepto por administradores del servicio destino. Este formato permite, en casos simple, el mantenimiento por el uso de editores de texto estándar, o en casos más complejos, el mantenimiento automático por el Sistema de Administración de Servicio Athena. [9] [10]

3.5.- Dominio Kerberos - Reinos

Venimos considerando el caso donde hay un simple AS y un simple TGS los cuales pueden o no residir en la misma máquina.

Cuando hay un grupo de computadoras el número de pedidos al AS y al TGS es mayor y se puede generar un cuello de botella en el proceso de autenticación.

Por tal motivo surge la necesidad de dividir las redes dentro de reinos, donde cada reino tiene su propio AS y TGS. Permitiendo que usuarios de un reino puedan acceder a servicios en otro reino.

Un ambiente Kerberos consiste de un Server Kerberos con un número de clientes y un número de servers de aplicación.

Al sub conjunto de usuarios y servers registrados con un Server de Autenticación particular se lo denomina Dominio o Reino (REALM).

Requerimientos del ambiente :

- El Server Kerberos debe tener los identificadores de usuarios y las passwords de los mismos en sus Bases de Datos. Todos los usuarios son registrados con el Server Kerberos.
- El Server Kerberos debe compartir una clave secreta con cada server ya que todos son registrados con el Server Kerberos.

Redes de clientes y servidores bajo diferentes organizaciones administrativas generalmente constituyen diferentes dominios. Los usuarios de un dominio pueden necesitar acceder a servers de otro dominio, y algunos servers pueden estar dispuestos a proveer servicios a usuarios autenticados de otros dominios.

Kerberos provee mecanismos de autenticación entre dominios.

Para dos dominios que soportan autenticación entre dominios se agrega el siguiente requerimiento:

- El server Kerberos de cada dominio comparte una clave secreta con el server del otro dominio.

Cómo hace un usuario que quiere utilizar un servicio que pertenece a otro reino?

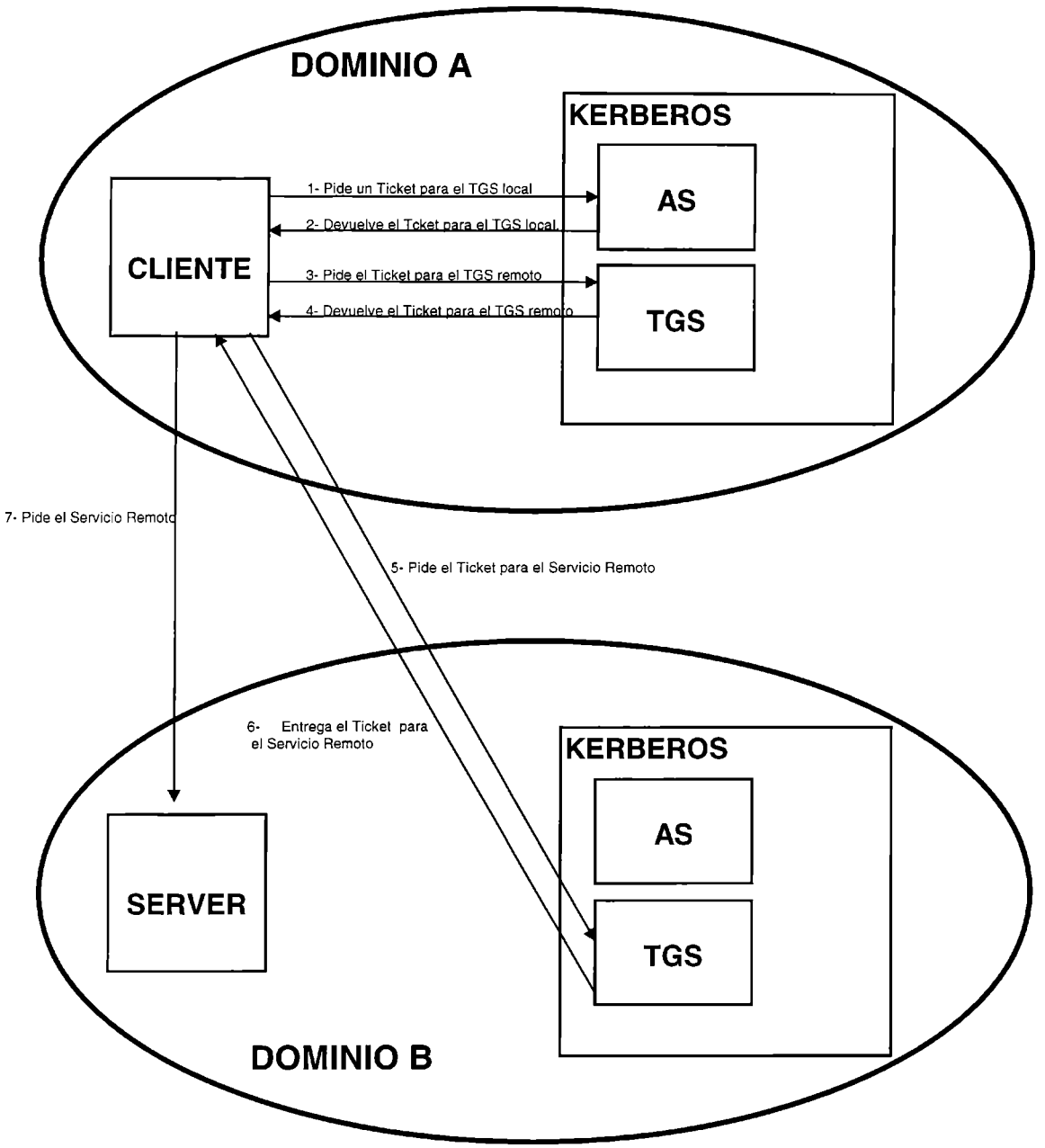
Primero, es necesario que el reino del usuario se registre al remoto TGS donde se encuentra el servicio que quiere utilizar el usuario. Entonces, el usuario pimero pide al AS el acceso al TGS, luego pide el acceso al remoto TGD (RTGS), y finalmente pide al RTGS el acceso al servicio remoto.

El esquema requiere que el Server Kerberos en un dominio confíe en el Server Kerberos en el otro dominio para autenticar sus usuarios, también los servidores participantes en el segundo dominio deberán confiar en el Server Kerberos del primer dominio.

Entonces cualquier usuario que desee el servicio de un server de otro dominio necesita un ticket para este server. El usuario sigue los procedimientos locales para acceder al TGS (ticket granting service) local y luego pide un Ticket Granting Ticket para un TGS remoto. Luego, el usuario puede suministrar al TGS remoto un ticket de servicio (Service Ticket Granting) para el server deseado en el dominio del server remoto.

El ticket presentado en el server remoto indica el dominio en el cual el usuario fue originariamente autenticado.

Un *problema* es que no es bueno tener varios dominios porque si hay N dominios se beberán hacer [N * (N-1)] / 2 intercambios de claves. Por ejemplo, en cuatro dominios se harán seis intercambios de claves. [1]



Detalles de los intercambios ilustrados en la figura anterior:

1) C → AS : ID_C || Dtgs || TS_1
2) AS → C : E_{KC} [$K_{C,TGS}$ | ID_{TGS} |TS_2 | tiempo de $vida_2$ | $Ticket_{TGS}$]
3) C → TGS : ID_{TGSrem} | $Ticket_{TGS}$ | $Autenticador_C$
4) TGS → C : $E_{K\,c,tgs}$ [$K_{C,TGS\,rem}$ | ID_{TGSrem} | TS_4 | $Ticket_{TGSrem}$]
5) C → TGS_{REM} : $ID_{V\,rem}$ | Ticket TGS_{REM} | Autenticador c
6) TGS → C : Ekc,TGS_{rem} [Kc,vrem | $ID_{V\,rem}$ | TS_6 | $Ticket_{V\,rem}$]
7) C → V_{REM} : $Ticket_{V\,rem}$ | $Autenticador_C$
V_{REM} : Server Remoto
ID_{TGSrem} : Identificador del TGS remoto.

3.6.- Componentes de Kerberos

La librería de aplicaciones Kerberos provee una interface para aplicaciones clientes y aplicaciones del servidor. Esta contiene, entre otras, rutinas para crear o leer pedidos de autenticación y rutinas para crear seguridad o mensajes privados.

Componentes del software de Kerberos:

- Librerías de aplicaciones Kerberos.
- Librerías de Encriptación.
- Librerías de Bases de Datos.
- Programas de administración de bases de datos.
- Administración del server.
- Autenticación del server.
- Software de propagación de la Base de Datos.
- Programas de usuario.
- Aplicaciones.

La encriptación en Kerberos está basada sobre DES (Data Encriptación Standard). La propuesta de DES es que el texto encriptado es desencriptado con la misma clave usada para encriptar.

3.6.1.- Administración de la Información Kerberos

La Base de Datos sobre la que se basa a Kerberos contiene un registro por cada identidad de usuario y por cada servicio (es decir, por cada servicio y/o usuario) conocido dentro del reino de Kerberos. Para permitir que la seguridad de la información sea consideración primaria cuando se realizan intercambios operativos en cuanto a la administración del servicio de Kerberos, la información que almacena es la mínima requerida para administrar la autenticación. Así, a pesar de que un registro de Kerberos es una especie de registro por usuario no contiene información como el número de teléfono o la dirección comercial, las cuales no son utilizadas por Kerberos para la autenticación. Sin embargo, si existe un gran número de usuarios, la Base de Datos Kerberos aún puede ser bastante grande y requiere algunas herramientas para su administración.

La interface de la administración de la información de Kerberos está diseñada para ser usada en dos formas:

- Por un conjunto de herramientas, manualmente desde una estación de trabajo del administrador del sistema. Este enfoque es favorable para la administración de un reino Kerberos que tenga un número pequeño de usuarios.
- Por un sistema de administración de servicios automatizado. Este enfoque está dirigido al manejo de un sistema con miles de usuarios. En ambos casos, la administración del servicio Kerberos se logra remotamente por medio de la red, usando conexiones seguras autenticadas de Kerberos. La

información almacenada por cada usuario/servidor que Kerberos está preparado para autenticar es la siguiente:

- El identificador del usuario/servidor incluyendo el identificador de instancia.
- La clave privada.
- La fecha de vencimiento para esta identidad.
- La fecha en la que el registro se modificó por última vez.
- La identidad del usuario/servidor que modificó este registro por última vez.
- El máximo tiempo de vida de los tickets.
- Los atributos (no usados).
- La implementación de la información no visible externamente.

3.6.2.- Replicación de la Base de Datos Kerberos

La Base de Datos Kerberos para un reino se administra y actualiza por un simple servidor de administración de Base de Datos Kerberos (KDBM), los pedidos de autenticación son manejados por uno o más servidores de distribución de claves Kerberos (KKDS), cada uno de los cuales contiene una copia idéntica de la Base de Datos. Ya que todos los KKDS tienen idéntica información, cualquier KKDS puede manejar cualquier pedido de autenticación; un cliente utiliza un nombre de servicio para obtener una lista de los KKDS y elegir el que le quede más cerca en términos de topología de red.

La separación de responsabilidad entre KDBM y los KKDS no implica que se requieran hosts diferentes; en un despliegue simple un host puede correr tanto en un server KDBM como en un KKDS. El propósito de la separación es simplificar la actualización de la base de datos mientras que le permite al KKDS mejorar su disponibilidad y performance (ya que muchos otros servicios de red dependen de él, la continua disponibilidad del servicio de distribución de claves es esencial; la continua disponibilidad de servicio de actualización no es en realidad tan importante).

Con respecto a la Base de Datos Kerberos todas las operaciones hechas por KKDS son *read-only* y por lo tanto, la única coordinación entre los KKDS y el KDBM es que el KKDS recibe actualizaciones de la información cuando se realizan cambios en el KDBM. De nuevo por simplicidad, el KDBM emite las actualizaciones del KKDS ocacionalmente y lo hace copiando toda la base de datos entera. La copia completa elimina la necesidad de tener procedimientos actualizados considerablemente más complejos que mantendrían las filas actualizadas en el KDBM y los procesos de recuperación en los KKDS. Debido a que las actualizaciones ocurren sobre una base bach, los KKDS pueden tener datos que estén apenas atrazados.

El KDBM copia su Base de Datos a los KKDS usando un protocolo protegido por Kerberos. Primero, usando el protocolo de autenticación mutua, se intercambia una clave encriptada segura entre el sitio de KDBM y el sitio de KKDS dado. El KDBM crea un punto de control de los datos a ser transferidos y calcula su fuerte *cheksum*, combinando cheksum con la clave de sesión, luego transfiere los datos reales usando un protocolo de transferencia de archivos convencional. Recuerde que los datos no incluyen ninguna password de texto claro u otra información particular sensible. Sin embargo, se debe asegurar su integridad su KKDS receptor almacena temporariamente todos los datos transferidos, luego recalculan el cheksum de los datos y los datos recibidos usando la clave de sesión secreta. Luego compara el *cheksum* calculado con el original el cual fue transmitido separadamente utilizando el protocolo de seguridad Kerberos. Si los dos

cheksum concuerdan, los datos recientemente recibidos actualiza la Base de Datos del KKDS.

Por lo tanto, podemos decir que Kerberos utiliza la librería de Base de Datos como herramienta para administrar la Base de Datos Kerberos.

La Base de Datos Kerberos contiene un registro para cada usuario y para cada servicio conocido dentro del dominio Kerberos.

Kerberos mantiene una Base de Datos de sus clientes y sus claves privadas. La clave privada es una palabra clave encriptada conocida sólo por Kerberos y el cliente a quien pertenece.

Kerberos también genera claves privadas temporarias llamadas claves de sesión. Una clave de sesión puede ser usada para encriptar mensajes entre dos partes.

La información que Kerberos almacena en su Base de Datos es la mínima requerida para efectuar la autenticación:

- El identificador del cliente
- La clave privada del cliente
- Datos de expiración para esa identidad.
- Máximo tiempo de vida del ticket para ser usado por el cliente
- Atributos.

El administrador del server (KDBM) provee una interface de red *read-write*, la parte cliente de un programa puede correr sobre cualquier máquina de la red. El server sin embargo debe correr sobre la máquina almacenando la base de datos Kerberos para hacerle cambios a la misma.

El Server de Autenticación o Server Kerberos, realiza operaciones sobre la Base de Datos Kerberos para autenticar los clientes y generar las claves de sesión, este server no modifica la base de datos Kerberos y puede correr sobre una máquina almacenando una copia de sólo lectura de la Base de Datos maestra Kerberos.

El software de programación de la Base de Datos maneja la replicación de la Base de Datos Kerberos.

Es posible tener copias idénticas de esta base en diferentes máquinas con una copia del server de autenticación corriendo sobre cada máquina para poder tomar y autenticar pedidos.

En cada máquina la Base de Datos Kerberos es *read-only*, cuando la base Kerberos cambia todas las copias son actualizadas mediante un protocolo Kerberos protegido.

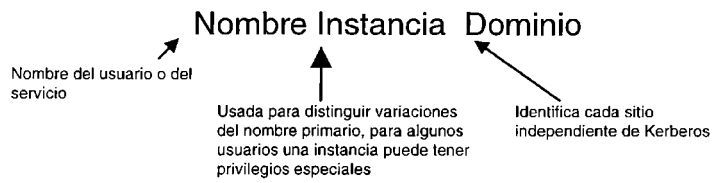
Las distintas rutinas de modificación hechas a la base son:

- Agregar un nuevo usuario.
- Cambiar la palabra clave de un usuario.
- El administrador del sistema cambia una clave olvidada o comprometida.
- Desactivar un viejo usuario.
- Remover los viejos identificadores de usuarios.

3.6.3.- Nombres de Kerberos

Para cada cliente y servicio Kerberos define un nombre principal único. Dicho nombre tanto para los clientes o servicios tiene la misma estructura, que consiste de un

Nombre Primario, una Instancia y un Dominio todo esto expresado de la siguiente manera:



Para servicios en ambiente Athena la instancia es usualmente el nombre de la máquina en el cual el server corre, por ejemplo, el servicio RLOGIN tiene diferentes instancias sobre diferentes hosts: RLOGIN.PRIM es el server RLOGIN sobre el host llamado PRIM. [3] [9] [10] [11]

3.7.- Kerberos Versión 5

3.7.1.- Diálogo de Autenticación de la Versión 5 de Kerberos

Vamos a detallar el diálogo básico de la versión 5, comparando con la versión 4.

Consideramos *authentication service exchange* (intercambio del servicio de autenticación).

El Mensaje 1 es un pedido del cliente por el TGT. Como anteriormente, incluye el ID del usuario y el TGS. Ahora se agregan los siguientes nuevos elementos:

- *Realm* : indica el reino del usuario
- *Options*: usados para pedir qie ciertos flags se seteen en el ticket retornado.
- *Times*: usados por el client para pedir los siguientes seteos de tiempos en el ticket
  - *-from:* el tiempo del comienzo deseado para el ticket pedido
  - *-till:* tiempo de expiración pedido por el ticket pedido.
  - *-rtime:* tiempo de renovación pedido
- *Nonce*: un valor aleatorio para ser repetido en el Mensaje (2) para asegurar que la respuesta es nueva y no ha sido usada nuevamente por un hackers

El Mensaje 2 retorna un t.g.t, identificando información para el cliente, y un bloque encriptado usando la clave de encriptación basada en la password del usuario. Este bloque incluye la clave de sesión a ser usada entre el cliente y el TGS, los tiempos especificados en el Mensaje 1, el *nonce* desde el mensaje 1, el TGS identificando información. El ticket mismo incluye la clave de sesión , identificando información para el cliente, los valores de tiempo pedido, y los flags que reflejan el estado del ticket y las opciones pedidas.

Estos flags introducen nuevas funcionalidades significativas a la versión 5.

Comparamos *ticket-granting service exchange* (el intercambio del t.g.s.) para la versión 4 y la versión 5. El mensaje 3 para ambas versiones incluye un autenticador, un ticket y el nombre del servicio pedido. Además, la versión 5 incluye tiempos pedidos y opciones para el ticket y un *nonce*, todas con funciones similares a aquel mensaje 1. El autenticador es esencialmente la misma como el usado en la versión 4.

El mensaje 4 tiene la misma estructura que el mensaje 2, retornando un ticket mas información necesaria por el cliente, encriptado luego con la clave de sesión ahora almacenado por el cliente y el TGS.

Finalmente, por el intercambio de autenticación cliente/servidor, varias nuevas características aparecen en la versión 5. En el mensaje 5, el cliente puede pedir como una opción que la autenticación mutua es requerida. El autenticador incluye varios nuevos campos como los siguientes:

- *Subkey* : la elige el cliente par una clave de encriptación a ser usada para proteger esta sesión de aplicación específica. Si este campo es omitido, la clave de sesión del ticket (Kc,v) es usada.
- *Sequence number:* campo opcional que especifica el número de secuencia de comienzo a ser usada por el servidor para mensajes que se envían al cliente durante esta sesión. Los mensajes pueden ser una secuencia numerada para detectar respuestas.

Si la autenticación mutua es requerida, el servidor responde el mensaje 6. Este mensaje incluye el *timestamp* desde el autenticador. Notar que en la versión 4, el *timestamp* fue incrementado en uno. Esto no es necesario en la versión 5 porque las características del formato del mensaje es tal que éste no es posible para un *hackers* crear el mensaje 6 sin conocimiento de las claves de encriptación apropiadas. El campo del número de secuencia opcional especifica el número de secuencia de comienzo a ser usado por el cliente

3.7.2.- Resumen del Intercambio de mensajes de la Versión 5 de Kerberos

a) Servicio de Autenticación para obtener un ticket granting ticket

(1) C → AS : Options | IDc | Realm c | IDtgs | Times | Nonce1
(2) AS → C : Realm c | IDc | Ticket tgs | Ekc[Kc,tgs | Times | Nonce1 | Realm tgs | IDtgs]
Ticket tgs : Ektgs[Flags | K c,tgs | Realm c | IDc | ADc | Times]

b) Ticket Granting Service para obtener el service granting ticket
(3) C → TGS : Opcions | IDv | Times | Nonce2 | Ticket tgs | Autenticador c
(4) TGS → C : Realm c | IDc | Ticket v | Ekc,tgs [Kc,v | Times | Nonce2 | Realm v | IDv]
Ticket tgs : Ektgs[Flags | K c,tgs | Realm c | IDc | ADc | Times]
Ticket v : Ekv[Flags | K c,v | Realm c | IDc | ADc | Times]
Autenticador c : Ekc,tgs [IDc | Realm c | TS1]

c) Cliente/Server , intercambio para obtener el servicio

(5) C → Server : Options | Ticket v | Autenticador c
(6) Server → C : Ekc,v [TS2 | Subkey | Seq#]
Ticket v : Ekv[Flags | K c,v | Realm c | IDc | ADc | Times]
Autenticador c : Ekc,v [IDc | Realm c | TS2 | Subkey | Seq#]

Los flags que pueden incluirse en un ticket son los que indican los tipos de tickets que a continuación detallamos:

- *Tickets iniciales y pre-autenticados:* indica que un ticket fue emitido usando protocolo AS, o sea que la clave del cliente ha sido recientemente presentada y no basado en un ticket proveedor de tickets (TGT).
- *Tickets no válidos:* los servidores deben rechazar los tickets que de alguna manera se identifican como no válidos. Un ticket posdateado sera emitido usualmente de esta forma. Los tickets no válidos deberán ser validados por el KDC.
- *Tickets Renovables:* Las aplicaciones pueden querer tener tickets que sean válidos por largos períodos de tiempo. Sin embargo esto puede exponerlos a robos y estos tickets robados serían válidos hasta la fecha de vencimiento del ticket. Simplemente usando tickets de corta vida y obteniendo nuevos períodicamente requeriría que el cliente tuviera que reentrar varias veces su password, un riesgo aún mayor. Los tickets renovables pueden usarse para mitigar las consecuencias del robo, tienen dos fechas de vencimiento, la primera es cuando vence la instancia actual del ticket y la segunda es el último valor permisible para una fecha de vencimiento individual, entonces cuando se vence la primer fecha el cliente presenta el ticket al KDC y este emitirá un nuevo ticket con una nueva clave de sesión y una fecha de vencimiento posterior, así hasta llegar a la última fecha permisible del ticket.
- *Tickets posdateados:* Las aplicaciones ocacionalmente pueden necesitar obtener tickets para su uso mucho más tarde, pero es peligroso retener tickets válidos ya que estarán en on_line por más tiempo y serán más propensos al robo. Los tickets posdateados proveen una forma de obtener estos tickets del KDC al momento de la realización del trabajo, se dejan en suspenso hasta que sean validados y activados por un futuro pedido del KDC entonces si se reporta el robo de un ticket en el interín el KDC rechazaría validar el ticket y el ladrón sería burlado.
- *Ticket proxy y proxiables:* A veces puede ser necesario para un principal permitirle a un servicio realizar una operación en representación suya otorgándole un proxy. El servicio debe ser capaz de obtener la identidad del cliente, pero solo para un propósito en particular. Por ejemplo: un cliente del servicio de impresión puede darle un proxy al servidor de impresión para acceder a los archivos del cliente en un servidor de archivos en particular para satisfacer su pedido de impresión.

3.7.3.- Diferencias entre la Versión 4 y la Versión 5

- Dependencia del sistema de encriptación : versión 4 requiere del uso de DES. En la versión 5, el texto es encriptado con un identificador del tipo de encriptación, entonces puede ser usada cualquier técnica de encriptación. Las claves de encriptación son rotuladas con un tipo y una longitud permitiendo que la misma clave sea usada en diferentes algoritmos .
- Dependencia del Protocolo de Internet (IP) : La versión 4 requiere del uso de direccionamiento IP. Otros tipos de direcciones tal como las direcciones ISO

networks no son adecuadas. En la versión 5 se pueden utilizar cualquier tipo de direccionamiento , los cuales son identificados con tipo y longitud.

- *Message byte ordening* : En la versión 4 no se establecieron convenciones para Message byte ordening. En la versión 5 todas las estructuras de mensajes fueron definidas usando Abstract Syntax Notation One (ASN.1) y Basic Encoding Rules (BER) que provee un ordenamiento de bytes no ambiguo.
- Tiempo de vida del ticket : En la versión 4 los valores del tiempo de vida son codificados en 8 bits cada 5 minutos. Entonces el máximo tiempo de vida que puede ser expresado es $2^8 * 5 = 1280$ minutos, aproximadamente 21 hs, esta puede ser inadecuada para algunas aplicaciones (por ejemplo en simulaciones de larga duración). . En versión 5 los tickets incluyen explícitamente el tiempo de inicio y de fin, permitiendo tickets con tiempos de vida arbitrarios.
- *Forwarding de autenticación:* la versión 4 no permite a un cliente forwardear su identidad a algún otro host y ser usada por algún otro cliente. Esta capacidad habilitaría a un cliente acceder a un servidor y que esta tenga acceso a otro servidor en beneficio del cliente. La version 5 provee esa capacidad.
- Autenticación entre reinos : para N reinos, Kerberos requiere n2 (n cuadrado!) relation-ship Kerberos-to-Kerberos. La versión 5 soporta un método que requiere pocas relaciones.

Además de las limitaciones de entorno, existen deficiencias técnicas en la versión 4 del protocolo:

- *Doble encriptación* : en la versión 4 los tickets provistos al cliente son encriptados dos veces una con la clave secreta del servidor destino y luego con una clave secreta conocida por el cliente. La segunda encriptación no es necesaria y computacionalmente no es económico.
- En la versión 4 hace uso de un modo no standard de encriptación DES, *Encriptación PCBC* (Plain-and-Cipher Block Chaining) . Este método es vulnerable a un ataque que involucre el intercambio de bloques de texto encriptado. La versión 5 provee mecanismos de integración explícito, permitiendo usar el modo CBC standard para la encriptación.
- *Claves de sesión:* la clave de sesión debe ser usada por el cliente y el servidor para proteger mensajes transmitidos durante esta sesión. De esta manera, como el mismo ticket debe ser usado repetidas veces para obtener el servicio desde un servidor particular, existe el riesgo que un adversario repita el mensaje desde una sesión vieja al cliente o al servidor. En la versión 5 es posible que el cliente y el servidor negocien una clave de sub sesión, la cual es usada solamente para una conexión. Un nuevo acceso del cliente resultaría en el uso de una nueva clave de subsesión.
- *Ataques a las passwords*: una vulnerabilidad compartida por ambas versiones es el ataques a las passwords. El mensaje desde el AS al cliente incluye información encriptada con una clave basada en la password del cliente. Un adversario puede capturar el mensaje y desencriptarlo probando varias passwords posibles. Si se descubre la password del cliente el

adversario podrá usarla posteriormente para obtener credenciales de autenticación de Kerberos. La versión 5 provee un mecanismo conocido como pre-autenticación, el cual podría hacer el ataque de password mas dificultoso, pero esto no lo previene totalmente. [1] [21]

3.8.- Problemas sin solución de Kerberos

Hay varios problemas asociados al mecanismo de Kerberos, en medio de las impresiones de los tickets está el problema de decidir el tiempo correcto de vida, como conceder los proxis y como garantizar la integridad de la estación de trabajo.

El problema del tiempo de vida del ticket es la manera de elegir entre seguridad y conveniencia, si el tiempo de vida del ticket es largo entonces si el ticket y su clave de sesión asociados son robados o extraviados ellos pueden ser usados durante un largo período de tiempo. Tal información puede ser robada si el usuario se olvida de salir de la estación de trabajo. Alternativamente si el usuario ha sido autenticado sobre un sistema que permite usuarios múltiples, otro usuario con acceso al root puede estar disponible para encontrar la información necesaria para usar tickets robados. El problema con un ticket con un corto tiempo de vida es que cuando este expira, el usuario tendría que obtener uno nuevo y para hacerlo debería entrar su password.

Un problema abierto es el *problema del proxy*. Cómo puede un usuario autenticado permitir actuar en otros servicios de red en nombre de él. Un ejemplo donde esto podría ser importante es el uso de un servicio que lograría acceder a los archivos protegidos directamente desde un file server. Otro ejemplo de este problema sería lo que llamamos *autenticación forwarding*, si un usuario está conectado en una estación de trabajo y se conecta en un host remoto sería bueno si el usuario tuvo acceso a los mismos servicios disponibles localmente, mientras corre un programa sobre el host remoto. Lo que hace esto dificultoso es que el usuario no debería confiar en el host remoto, así la autenticación forwarding no es deseada en todos los casos. No tenemos en la actualidad una solución a este problema.

Otro problema, que es importante en el ambiente Athena, es como garantizar la integridad del software corriendo sobre una estación de trabajo. Este no es tanto un problema en una estación de trabajo privada ya que el usuario que lo estaría usando tiene el control sobre él. En las estaciones de trabajo públicas, sin embargo, alguien tiene que seguir y modificar el programa Login para salvar la password del usuario. La única solución actualmente disponible en nuestro ambiente es hacer que sea difícil para las personas modificar el software que corre en las estación de trabajo públicas. Una solución mejor requiere que la clave del usuario nunca deje el sistema el cual el usuario sabe que es confiable. Una manera en que esto podría hacerse sería si el usuario tuviera una Start Card (tarjeta inteligente) capáz de hacer las encriptaciones requeridas en el protocolo de autenticación.

3.9.- Diferentes ataques a Kerberos

Impersonal A

Un *hackers* C, podría robar el autenticador y el ticket mientras es transmitido a través de la red y los usa para despersonalizar a A.

La dirección en el ticket y el autenticador fue agregado para hacer mas difícil la performance de este ataque.

Para tener éxito C tendrá que usar la misma máquina que A o falsificar la dirección fuente de los paquetes. Incluyendo el *timestamp* en el autenticador, C no tiene mucho tiempo montar el ataque.

Impersonal B

C puede enmascarar la dirección de red de B, y cuando A envía sus credenciales, C solo pretende verificarlas. C no puede estar seguro de que ella esta hablando con A.

3.10.- Estrategias de Defensa

Sería posible agregar una *'replay cache'* del lado del server. La idea es salvar los autenticadores enviados durante los últimos pocos minutos, para que B pueda detectar cuando alguien está tratando de retransmitir un mensaje ya usado. Esto es algo poco práctico (en gran parte, relacionado a la eficiencia), y no es parte de Kerberos 4; MIT Kerberos 5 lo contiene.

Para autenticar B, A puede decir que B envíe algo de retorno (como respuesta) que pruebe que B tiene acceso a la clave de sesión. Un ejemplo de esto es el *checksum* que A envió como parte parte del autenticador. Un procedimiento típico es agregar un 1 al *checksum*, encriptarlo con la clave de sesión y enviarlo devuelta a A. Esto es llamado autenticación mutua.

La clave de sesión puede también ser usada para agregar *checksum criptograficos* a los mensajes enviados entre A y B (conocido como integridad de mensajes). También se puede agregar la Encriptación (confidencialidad de mensajes). Esto probablemete es la mejor aproximación en todos los casos.

3.11.- Fallas de Seguridad en Kerberos

Un informe del Wall Street Journal del febrero de 1996 destacó que una falla de seguridad mayor fue descubierta en un sistema de seguridad Internet usando Kerberos por dos estudiantes de la Univesidad de Purdue. El informe dijo que la falla encontrada por los estudiantes podría permitirle a un *hackers* penetrar en la red de empresas en alrededor de 5.8 segundos. El informe indicó que el *hackers* podría leer el mail confidencial, tener acceso a los archivos privados bajo ciertas circunstancias y camuflarse como un usuario válido.

La falla, primero fue descubierta por un estudiante graduado de Purdue, Steven Lodin. Cuando en octubre de 1995, las noticias del descubrimiento de una falla en el software de Netscape de dos estudiantes de Berkley salieron a la luz. Con ayuda de estudiantes graduados, Lodin fue capaz de meterse en el software de Kerberos. Como en la falla de Netscape, la *password aleatoria* (o clave de sesión en KERBEROS) ya no es más tan al azar. Versiones anteriores de Kerberos crearon la clave usando una selección mas pequeña de números de alrededor de un millón de posibilidades, haciéndole mas fácil a las computadores adivinar la clave. 'Una vez que vos sepas que esta ahí, es realmente trivial explotarla', dijo Eugene Spafford, profesor asociado en ciencias de computación en Purdue, donde llevo a un estudiante una tarde a abrir efectivamente la traba (*lock*) del software.

Las computadoras y estación de trabajo más poderosas de hoy, son capaces de realizar muchos más cálculos para violar el código que las máquinas de hace 10 años cuando Kerberos fue diseñado por primera vez. En agosto del 95, un estudiante francés del Ecole Polytechnique usó más de 100 estaciones de trabajo de alta performance para violar la clave de seguridad que el gobierno de los Estados Unidos (US) requiere que la Corporación de Comunicaciones Netscape use en sus software de browsers de Internet. Meses más tarde, los estudiantes de Berkley descubrieron una nueva falla en el software Netscape, una acción que impulsó a la compañía a instituir un programa que le otorgue a los descubridores de la falla de seguridad un premio en efectivo.

Jeffrey Schiller, uno de los desarrolladores de Kerberos y gerente de la red en el MIT, reconoció la falla pero dijo que si aún fuera corregida, el parámetro de gobierno sobre el cual estaba basado está considerando como 'riesgosa' la fuerza y el más bajo costo de las computadoras de alta performance de hoy y dijo que 'hace diez años, los chicos malos no podían acceder al mismo tipo de computadoras que las que un banco podía' . Schiller dijo que los ingenieros de software del MIT han sabido desde 1986 acerca del problema que existió en Kerberos V4, y lo repararon en la versión subsiguiente. La versión 4 aún es usada ampliamente en todo el mundo, ya que el MIT provee el software libremente. [23]

3.12.- Conclusiones

Por más de 10 años hasta ahora, los gerentes de sistemas de computadoras han confiado en el sistema de autenticación Kerberos para proteger la integridad de las redes. Aún popular en la actualidad, puede ser encontrado en los últimos browsers de la web usados por muchos para buscar en Internet. Los tickets encriptados de Kerberos y la tecnología de paso de clave ha alejado a muchos impostores de la violación de la seguridad de algunos de los datos más sensibles del mundo. A medida las redes crecen más distribuídas , crece la necesidad de seguridad. Quedan preguntas sobre por cuánto tiempo más Kerberos puede crecer con ellas para permanecer como un método de seguridad posible, como se noto con anterioridad la mejora continua en el rendimiento y capacidades de las computadoras , especialmente de las computadoras accesibles. Este motivo esta presionando a la gente de la seguridad de la red para reconsiderar la efectividad total de Kerberos. Las versiones V4 ya no son seguras , y es solo cuestión de tiempo antes de que alguien innove con la versión V5.

Es necesario algún otro método para las computadoras para que identifiquen positivamente quién está tratando de conectarse o de pedir servicios?

Con la explosión de la conectividad a nivel mundial traída por Internet o por la WWW, las computadoras de todo el mundo tienen acceso a millones de otras computadoras. La habilidad de Kerberos para proveer un método de autenticación sin enviar el código a través de la red fue un gran avance diez años atrás, pero Internet no era más que un puñado de computadoras educativas unidas todas juntas al mismo tiempo.

Más alla de la migración de poderosas estaciones de trabajo y computadoras personales, una forma popular de computadoras puede estar también en el horizonte. Las computadoras de la red las cuales ofrecerán acceso a redes distribuídas con muy poco poder de procesamiento interno, podrían ser la próxima opción para el hogar o la oficina. Estas nuevas terminales *'tontas'* podrían ofrecer acceso efectivo de costo a las redes como Internet o redes corporativas pero quizás no tenga la capacidad de correr un

software Kerberos, así limitando los niveles de acceso de seguridad posibles allí. Las estaciones de trabajo poderosas aún existirán, y podrían fácilmente hacer estragos en las redes diseñadas para el acceso a las NC.

Por ahora, Kerberos V5 es aún un método altamente confiable para brindar autenticación en las redes de computadoras distribuidas de la actualidad. A pesar de que su vida útil permanece cuestionada.

Capítulo 4

Lenguaje JAVA

4.1.- Introducción al lenguaje

Java es un lenguaje de programación desarrollado por *Sun,* que combina tres elementos claves para conseguir que sea una tecnología completamente diferente de lo que existe hoy en día. En primer lugar, y el más obvio es la capacidad de que cualquiera pude por primera vez utilizar los applets, que son aplicaciones pequeñas, seguras, dinámicas, multiplataforma, activas y en red. Entonces un applet es un programa pequeño en Java que se transfiere dinámicamente a través de la red, igual que una imagen, archivo de sonido o animación. La diferencia fundamental que un applet es un programa inteligente que puede reaccionar a la entrada de un usuario y cambiar dinámicamente. Los applets son componentes de software independientes, que son bajados en formato binario de bytecode separados de la página Web que les hace referencia. Cuando llegan a la máquina cliente, los applets pueden ser interpretados por el browser en un ambiente seguro elegido por el browser. En segundo lugar Java es un lenguaje de programación poderoso que ofrece la potencia del diseño orientado a objetos con una sintaxis simple y familiar, en un entorno robusto y agradable de utilizar. En tercer lugar Java es un rico conjunto de clases de objeto poderosas que proporcionan al programador abstracciones claras para muchas funciones del sistema habituales, como la gestión de ventanas, de red y de entrada/salida.

Este lenguaje es distribuido y de arquitectura neutral, soporta la programación de aplicaciones Web en forma de Java applets de plataforma independiente.

Siendo un lenguaje de propósito general, Java tiene acceso a un rico conjunto de librerías de clases que soportan el acceso a manejo de archivos, protocolos de red, manipulación directa, interfaz de usuario gráfica, etc. Esto lo hace ideal para aplicaciones Web, el cliente tiene un conjunto de applets con gran capacidad de procesamiento y puede implementar un protocolo de red conveniente para proveer una interacción adecuada con el servidor. Esto hace que los applets reemplacen a los CGI scripts en su totalidad. Un acierto de los applets de Java es que pueden distribuir los tiempos de respuesta y la funcionalidad de la interfaz de usuario al igual que las aplicaciones corriendo localmente sobre su sistema operativo estándar. Una tendencia creciente es proveer applet de propósito general los cuales pueden ser parametrizados cuando se colocan dentro de una página Web.

Java tiene ciertas características, muy importantes, que lo hacen un lenguaje:

- *Simple*

  Para poder programar en Java se deberán conocer los conceptos de la programación orientada a objetos ya que da la posibilidad de expresar toda idea que tenga de una manera orientada a objetos directa y limpia. En este lenguaje hay un número reducido de maneras de realizar una tarea dada.

Java es similar a C++, pero resulta mucho más simple. No tiene sobrecarga de operadores, archivos de encabezado, preprocesadores, uniones, estructuras, arreglos multidimensionales y conversión de tipo implícita. Además, los programadores ya no deben preocuparse acerca de la administración de la memoria, pues Java incorpora un programa denominado "Garbage Collector", que hace un scanning de la memoria y libera automáticamente cualquier pieza de memoria que ya no está siendo utilizada.

- *Orientado a Objetos*

  Java es un lenguaje de programación orientado a objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo, y tal como sucede en todo lenguaje orientado a objetos, el código de Java está organizado en clases. Cada clase define el comportamiento de un objeto. Una clase puede heredar conductas desde otra clase diferente. En la raíz de la jerarquía de clases, siempre está la clase Objeto.

  Soporta una jerarquía de clases con una sola herencia. Esto significa que cada clase solamente puede heredar desde una clase a la vez.

  Java también soporta interfaces, las cuales son en realidad clases abstractas. Esto permite que los programadores puedan definir métodos para interfaces sin tener que preocuparse inmediatamente en cómo se implementarán dichos métodos. Una clase puede implementar múltiples interfaces.

- *Distribuido*

  Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como *http* y *ftp*.

  Las aplicaciones Java pueden abrir y acceder a los objetos a través de Internet (vía las URL's), tan fácil como se puede acceder a archivos locales.

- *Interpretado y compilado*

  Java es independiente de la plataforma porque cuando compila pasa a una representación intermedia llamada código de bytes que se puede interpretar en cualquier sistema que tenga un intérprete de Java.

  Al ejecutarse un programa Java, este es en primer término compilado a código en bytes. Estos *bytecodes* son similares a instrucciones de máquina lo cual hace que los programas Java lleguen a ser muy eficientes. No son específicos en ninguna máquina en particular, por lo cual pueden ser ejecutados en varias y diferentes computadoras sin tener que ser recompilados.

- *Robusto*

  Verifica su código a medida que lo escribe y una vez más antes de ejecutarlo para asegurarse. Debido a que Java es un lenguaje muy estricto en cuanto a tipos y declaraciones, la mayoría de los errores se pueden descubrir durante la compilación.

  El sistema Java controla cuidadosamente cada acceso a memoria y se asegura que el mismo sea legal. Si sucede algo imprevisto, lanzan un código de excepción. Los programas han de identificar dichas excepciones y las manejarán.

- *Seguro*

  El lenguaje Java en sí mismo fue diseñado para ser seguro, y su compilador garantiza que el código fuente no pueda violar las normas de seguridad. Los programas Java no cuentan con punteros, y además, es un lenguaje fuertemente tipado, de modo que resulte posible verificar estos programas antes de su ejecución.

  Los bytecodes ejecutados por el motor runtime de Java son verificados para resguardar que también ellos obedezcan a dichas reglas. Esta capa protege contra la posibilidad de que un compilador alterado pueda producir código que viola las reglas de seguridad.

  El Alimentador de Clases garantiza que las clases no violen el espacio de nombres o restricciones de acceso en el momento en que son cargadas dentro del sistema.

  Finalmente, la seguridad específica para API's evita que los Applets realicen acciones destructivas.

- *Multihilado*

  Un Programa Java puede tener más de un hilo de ejecución, asi los usuarios no tienen que detener su trabajo para esperar que Java complete operaciones prolongadas.

  Java provee dispositivos destinados a la sincronización que resultan muy sencillos para su uso, lo que hace que la programación sea mucho más amigable.

- *Portable*

  El lenguaje Java es el mismo en cualquier computadora. Portar programas Java es muy fácil ya que no necesitan ser recompilados. Las librerías definen interfaces portables.

- *Independiente de la Plataforma*

  El código compilado es ejecutable en cualquier procesador dado la presencia del sistema runtime de Java. El compilador Java realiza esta conversión, generando instrucciones de bytecode, los cuales no dependen de ninguna arquitectura de computadora en particular. Estos bytecodes están diseñados para ser fácilmente interpretados y traducidos a código nativo de máquina.

- *Extensible*

  Es posible hacer una interface del programa Java a las librerías de software existentes, y que estén escritas en otro lenguaje.

  Todos los programas Java corren dentro de un buen browser, pero es posible y útil escribir programas stand alone, que corran independientemente de un browser. Estos programas, generalmente llamados aplicaciones, son completamente portables. Sólo toman el código y lo corren en otra máquina.

- *Interactivo*

  Java fue diseñado para cumplir el requisito del mundo real de crear programas de red interactivos . El intérprete de Java viene con la solución más elegante diseñada hasta ahora para sincronización entre múltiples procesos, y hace que sea posible construir sistemas interactivos que se ejecuten sin problema.

4.2.- Conceptos del lenguaje JAVA

Un archivo fuente de Java llamado comunmente unidad de compilación, es un archivo de texto que contiene una o más definiciones de clase. Estos archivos se almacenan con la extensión de archivo .Java. Cuando se compila el código fuente de Java, cada clase individual se coloca en un archivo de salida propio con el nombre de la clase con una extensión .class.

Los programas Java se construyen con clases. A partir de una clase podemos crear cualquier cantidad de objetos o instancias de esa clase.

Las clases están constituidas por variables y métodos, las variables de datos están asociadas con una clase y sus objetos; los métodos contienen el código ejecutable de una clase.

Cuando se programa en Java, se coloca todo el código en métodos, de la misma forma que se escriben funciones de lenguajes como C.

Luego ampliaremos el concepto de clases y métodos.

El lenguaje Java se compone de:

- *Identificadores*
  Los identificadores nombran variables, funciones, clases y objetos, cualquier cosa que el programador necesite identificar o usar.
  En Java, un identificador comienza con una letra, un subrayado (_) o un símbolo de dólar ($). Los siguientes caracteres pueden ser letras o dígitos. Se distinguen las mayúsculas de las minúsculas y no hay longitud máxima.
- *Palabras clave*
  Las siguientes son algunas de las palabras claves que están definidas en Java y que no se pueden utilizar como indentificadores: Abstract, New, public, int, Void, etc.
- *Palabras Reservadas*
  Entre otras mencionamos cast, operator, future,rest, generi, var
- *Literales*
  Un valor constante en Java se crea utilizando una representación literal de él. Java utiliza cinco tipos de elementos: enteros, reales en coma flotante, booleanos, caracteres y cadenas, que se pueden poner en cualquier lugar del código fuente de Java. Cada uno de estos literales tiene un tipo correspondiente asociado con él.
- *Operadores*
  Los operadores de Java son muy parecidos en estilo y funcionamiento a los de C. Entre otros tenemos el and (&) el or (|) y el not (¡).
- *Tipos simples*
  El rendimiento del sistema fue una meta fundamental en el desarrollo de Java, esta decisión condujo a la creación de los tipos simples de Java. No estan en absoluto orientado a objetos, y son análogos a la mayoría de los otros lenguajes que no utilizan objetos. Los tipos simples representan expresiones atómicas de un solo valor. Java tiene ocho tipos simples que son: byte, short, int, long, char, float, boolean y double. [5] [13]

4.3.- Control de Flujo

Muchas de las sentencias de control del flujo del programa se han tomado del C:

- *Sentencias de Salto*

  - if/else

```
if( Boolean ) {
   sentencias;
   }
else {
   sentencias;
}
```

  - switch

```
switch( expr1 ) {
   case expr2:
      sentencias;
      break;
   case expr3:
      sentencias;
      break;
   default:
      sentencias;
      break;
   }
```

- *Sentencias de Bucle*

  - Bucles for

```
for( expr1 inicio; expr2 test; expr3 incremento ) {
   sentencias;
   }
```

    También se soporta el operador coma (,) en los bucles for

```
for( a=0,b=0; a < 7; a++,b+=2 )
```

  - Bucles while

```
while( Boolean ) {
   sentencias;
   }
```

  - Bucles do/while

```
do {
   sentencias;
   }while( Boolean );
```

- *Excepciones*

  - try-catch-throw

```
try {
   sentencias;
} catch( Exception ) {
   sentencias;
}
```

Java implementa excepciones para facilitar la construcción de código robusto. Cuando ocurre un error en un programa, el código que encuentra el error lanza una excepción, que se puede capturar y recuperarse de ella. Java proporciona muchas excepciones predefinidas.

4.4.- Clases

El elemento básico de la programación orientada a objetos en Java es la clase, esta define la forma y comportamiento de un objeto. Cualquier concepto que se desee representar en un programa Java está encapsulado en una clase.

Para crear una clase solo se necesita un archivo fuente que contenga la palabra clave class seguida de un identificador legal y un par de llaves para el cuerpo.

Una clase en Java puede contener variables y métodos. Las variables pueden ser tipos primitivos como int, char, etc.

Los métodos son funciones.

Por ejemplo, en el siguiente trozo de código podemos observarlo:

```
public MiClase {
   int i;
   public MiClase() {
      i = 10;
      }
   public void Suma_a_i( int j ) {
      i = i + j;
      }
   }
```

La clase MiClase contiene una variable (i) y dos métodos, MiClase que es el constructor de la clase y Suma_a_i(int j).

Las clases se componen por métodos que son funciones que pueden ser llamadas dentro de la clase o por otras clases. El constructor es un tipo específico de método que siempre tiene el mismo nombre que la clase.

Cuando se declara una clase en Java, se pueden declarar uno o más constructores opcionales que realizan la inicialización cuando se instancia (se crea una ocurrencia) un objeto de dicha clase.

Utilizando el código de ejemplo anterior, cuando se crea una nueva instancia de MiClase, se crean (instancian) todos los métodos y variables, y se llama al constructor de la clase:

```
MiClase mc;
mc = new MiClase();
```

La palabra clave new se usa para crear una instancia de la clase. Antes de ser instanciada con new no consume memoria, simplemente es una declaración de tipo. Después de ser instanciado un nuevo objeto mc, el valor de i en el objeto mc será igual a 10. Se puede referenciar la variable de instancia i con el nombre del objeto:

```
mc.i++; // incrementa la instancia de i de mc
```

Al tener mc todas las variables y métodos de MiClase, se puede usar la primera sintaxis para llamar al método Suma_a_i() utilizando el nuevo nombre de clase mc:

```
mc.Suma_a_i( 10 );
```

y ahora la variable mc.i vale 21.

Como dijimos anteriormente las clases se componen por métodos, que son la interfaz funcional de una clase. Se declaran dentro de la clase al mismo nivel que las variables de instancia. Se deben llamar a los métodos en el contexto de una instancia concreta de esa clase. La forma general de declarar un método es la siguiente:

```
Tipo nombre_del_método (lista de parámetros) {
   cuerpo del método
   }
```

Luego el método se llama dentro de una instancia de la clase utilizando el operador punto (.), la forma general de llamar a un método es la siguiente:

```
Referencia_a_objeto.nombre_del_método(lista_de_parámetros);
```

Puede ser tedioso inicializar todas las variables de una clase cada vez que se crea una instancia, por esta razón las clases implementan un método especial llamado constructor que inicializa un objeto inmediatamente después de su creación., tienen exactamente el mismo nombre de la clase en la que residen y devuelven el tipo de la clase explícitamente. Se llama al método constructor justo después de crear la instancia y antes de que new vuelva al punto de la llamada.

Todos los métodos y variables de instancia se pueden sobreescribir por defecto, para que ninguna subclase lo sobreescriba hay que declararlos como *final*. Este modificador implica que todas las referencias futuras a este elemento se basarán en esta definición.

Cuando se desea crear un método que se utiliza fuera del contexto de cualquier instancia se lo deberá declarar como *static* , las variables también se pueden declarar como static pero deberá ser consiente que será como declararlas como variables globales.

Hay situaciones en las que se necesita definir una clase que declara la estructura de una abstracción dada sin proporcionar una implementación completa de cada método. Puede pasar que se necesite que los métodos se sobreescriban en las subclases utilizando el modificador de tipo *abstract* . Cualquier clase que contenga métodos declarador como abstract también deberá declararse como abstract. No se pueden crear instancias de la clase con el operador new dado que su implementación completa no está disponible.

Una de las principales ventajas de la orientación a objetos es la capacidad para extender el comportamiento de una clase existente.

La Herencia es el mecanismo por el que se crean nuevos objetos definidos en términos de objetos ya existentes. Por ejemplo, si se tiene la clase Ave, se puede crear la subclase Pato, que es una especialización de Ave.

```
class Pato extends Ave {
   int numero_de_patas;
   }
```

La palabra clave extends se usa para generar una subclase (especialización) de un objeto. Una Pato es una subclase de Ave. Cualquier cosa que contenga la definición de Ave será copiada a la clase Pato, además, en Pato se pueden definir sus propios métodos y variables de instancia. Se dice que Pato deriva o hereda de Ave.

Además, se pueden sustituir los métodos proporcionados por la clase base. Utilizando nuestro anterior ejemplo de MiClase, aquí hay un ejemplo de una clase derivada sustituyendo a la función Suma_a_i():

```
import MiClase;
public class MiNuevaClase extends MiClase {
   public void Suma_a_i( int j ) {
      i = i + ( j/2 );
      }
   }
```

Ahora cuando se crea una instancia de MiNuevaClase, el valor de i también se inicializa a 10, pero la llamada al método Suma_a_i() produce un resultado diferente:

```
MiNuevaClase mnc;
mnc = new MiNuevaClase();
mnc.Suma_a_i( 10 );
```

En Java no se puede hacer herencia múltiple, siempre se hereda desde una sola clase.

4.4.1.- Principales clases JAVA

- *Math*

  La clase Math representa la librería matemática de Java. El constructor de la clase es privado, por los que no se pueden crear instancias de la

clase. Sin embargo, Math es *public* para que se pueda llamar desde cualquier sitio y *static* para que no haya que inicializarla.
Si se importa la clase, se tiene acceso al conjunto de funciones matemáticas.

- *Character*
  Al trabajar con caracteres se necesitan muchas funciones de comprobación y traslación. Estas funciones están empleadas en la clase Character. De esta clase sí que se pueden crear instancias, al contrario que sucede con la clase Math.
- *Float*
  El tipo *float* tiene el objeto *Float*. De la misma forma que con la clase Character, se han codificado muchas funciones útiles dentro de los métodos de la clase Float.
- *Double*
  El tipo *double* tiene el objeto *Double*. Hay muchas funciones útiles dentro de los métodos de la clase Double.
- *Integer*
  Cada tipo numérico tiene su propia clase de objetos. Así el tipo *int* tiene el objeto *Integer* con métodos de la clase Integer.
- *Long*
  El tipo *long* tiene el objeto *Long*. De la misma forma que con la clase Character, se han codificado muchas funciones útiles dentro de los métodos de la clase Long.
- *Boolean*
  Los valores *boolean* también tienen su tipo asociado *Boolean*, aunque en este caso hay menos métodos implementados que para el resto de las clases numéricas.
- *String y String Buffer*
  Java posee gran capacidad para el manejo de cadenas dentro de sus clases String y StringBuffer. Un objeto *String* representa una cadena alfanumérica de un valor constante que no puede ser cambiada después de haber sido creada. Un objeto *StringBuffer* representa una cadena cuyo tamaño puede variar.
  Los Strings son objetos constantes y por lo tanto muy baratos para el sistema. La mayoría de las funciones relacionadas con cadenas esperan valores String como argumentos y devuelven valores String.
  La clase StringBuffer dispone de muchos métodos para modificar el contenido de los objetos StringBuffer. Si el contenido de una cadena va a ser modificado en un programa, habrá que sacrificar el uso de objetos String en beneficio de StringBuffer, que aunque consumen más recursos del sistema, permiten ese tipo de manipulaciones.

4.5.- Interfaces

A veces se necesita solo declarar los métodos que un objeto debe soportar, sin facilitar la implementación de estos métodos. Java permite definir una interfaz que es como una clase pero solo con las declaraciones de sus métodos. Están diseñadas para admitir resolución de método dinámica durante la ejecución. Las clases pueden

implementar varias interfaces, para implementar una interfaz lo único que la clase necesita es la implementación del conjunto completo de métodos de la interfaz.

La forma general de una interfaz es:

```
Interface nombre {
Tipo_devuelto nombre_del_metodo1 (lista_de_parámetros);
Tipo_devuelto nombre_del_metodo2 (lista_de_parámetros);
  }
```

Todos los métodos que están implementando interfaces se deben declarar como públicos.

Una clase que implementa una interfaz tiene la siguiente forma:

```
Class nombre_clase implements nombre_interface {
   public tipo nombre_del_método (lista de parámetros) {
         cuerpo_del_método; }

  }
```

Las interfaces también pueden extender usando la palabra clave extends.

4.6.- Paquetes

La palabra clave package permite agrupar clases e interfaces. Los paquetes reciben un nombre y se pueden importar. Los nombres de los paquetes son palabras separadas por puntos y se almacenan en directorios que coinciden con esos nombres.

Por ejemplo, los ficheros siguientes, que contienen código fuente Java:
Applet.Java, AppletContext.Java, AppletStub.Java, AudioClip.Java
contienen en su código la línea:

```
package Java.applet;
```

Y las clases que se obtienen de la compilación de los ficheros anteriores, se encuentran con el nombre nombre_de_clase.class, en el directorio Java/applet

Los paquetes de clases se cargan con la palabra clave *import*, especificando el nombre del paquete como ruta y nombre de clase (es lo mismo que #include de C/C++). Cuando se necesita usar varias clases de un paquete puede importárselo completamente reemplazando el nombre por un "*".

```
import Java.Date;
import Java.awt.*;
```

Si un fichero fuente Java no contiene ningún package, se coloca en el paquete por defecto sin nombre. Es decir, en el mismo directorio que el fichero fuente, y la clase puede ser cargada con la sentencia import:

```
import MiClase;
```

4.6.1.- Principales paquetes de JAVA

El lenguaje Java proporciona una serie de paquetes que incluyen ventanas, utilidades, un sistema de entrada/salida general, herramientas y comunicaciones. Algunos de los paquetes Java que se incluyen son:

- *Java.applet*
  Este paquete contiene clases diseñadas para usar con applets. Hay una clase Applet y tres interfaces: AppletContext, AppletStub y AudioClip.
- *Java.awt*
  El paquete Abstract Windowing Toolkit (awt) contiene clases para generar widgets y componentes GUI (Interfaz Gráfico de Usuario). Incluye las clases Button, Checkbox, Choice, Component, Graphics, Menu, Panel, TextArea y TextField.
- *Java.io*
  El paquete de entrada/salida contiene las clases de acceso a ficheros: FileInputStream, FileOutputStream, ByteArrayInputStream y ByteArrayOutputStream.
- *Java.lang*
  Este paquete incluye las clases del lenguaje Java propiamente dicho: Object, Thread, Exception, System, Integer, Float, Math, String, etc.
- *Java.net*
  Este paquete da soporte a las conexiones del protocolo TCP/IP y, además, incluye las clases Socket, URL y URLConnection.
- *Java.util*
  Este paquete es una miscelánea de clases útiles para muchas cosas en programación. Se incluyen, entre otras, Date (fecha), Dictionary (diccionario), Random (números aleatorios) y Stack (pila FIFO).

4.7.- Protección de Acceso

Java proporciona muchos niveles de protección para permitir un control preciso de la visibilidad de las variables y métodos. Las clases y los paquetes son dos medios para encapsular y contener el ámbito de las variables y métodos. Los paquetes actuan como recipietes de clases y las clases como recipientes de código

Es obvio que dentro de una clase todas las variables y métodos son visibles para todas las otras partes de la misma clase, dado que la clase es la unidad de abstracción más pequeña en Java. Pero por la existencia de paquetes Java debe distinguir categorías de visibilidad entre los elementos de la clase de acuerdo a tres palabras claves : public, private y protected.

Cualquier cosa declarada como public se puede ver desde cualquier sitio. Cualquier cosa declarada como private no se puede ver desde el exterior de una clase. Si un elemento no tiene ningún modificador entonces será visible en las subclases a demás de en otras clases del mismo paquete. Cualquier cosa declarada como protected la verán las clases de otros paquetes que sean subclases directas de la clase de este paquete. Cualquier cosa declarada como private protected la verán todas las subclases independientemente del paquete en que estén.

La forma de declarar cualquiera de los accesos mencionados es la siguiente:

- *public*

  ```
  public void CualquieraPuedeAcceder(){}
  ```

  Cualquier clase desde cualquier lugar puede acceder a las variables y métodos de instacia públicos.
- *protected*

  ```
  protected void SoloSubClases(){}
  ```

  Sólo las subclases de la clase y nadie más puede acceder a las variables y métodos de instancia protegidos.
- *private*

  ```
  private String NumeroDelCarnetDeIdentidad;
  ```

  Las variables y métodos de instancia privados sólo pueden ser accedidos desde dentro de la clase. No son accesibles desde las subclases.
- *Sin modificador*

  ```
  void MetodoDeMiPaquete(){}
  ```

  Por defecto, si no se especifica el control de acceso, las variables y métodos de instancia se declaran sin modificador, lo que significa que son accesibles por todos los objetos dentro del mismo paquete, pero no por los externos al paquete. [5]

4.8.- Applets y Aplicaciones

Un applet es una pequeña aplicación accesible en un servidor Internet, que se transporta por la red, se instala automáticamente y se ejecuta en el lugar como parte de un documento web. Están basados en un formato gráfico sin representación independiente. Es el elemento interactivo a embeber en otras aplicaciones.

Una aplicación es un programa Java stand alone que corre independientemente de un browser. Las aplicaciones son completamente portables ya que solo toman el código y lo corren en otra máquina.

Desde el punto de vista del programador la gran diferencia entre un Applet y una aplicación es que los primeros no tienen el método main() , u otro método por el cual se inicia la ejecución.

Los applet no tienen el control de la ejecución, simplemente responden cuando el navegador les avisa.

Componentes de un applet:

El lenguaje Java implementa un modelo de programación orientado a objetos. Los objetos sirven de bloques centrales de construcción de los programas Java. También tiene variables de instancia y métodos.

4.8.1.- Estructura de un Applet

```
/*Sección de importaciones */
public class NombreDelNuevoApplet extends Applet {
/*declaración de las variables de estado y métodos*/
/*declaración de métodos para la interacción con los objetos*/
public void MetodoUno( parámetros ) {
 /*Código Java que desempeña la tarea.*/
      }
}
```

4.8.2.- Métodos de la clase Applet

Se utilizan para iniciar y detener la ejecución del Applet, para pintar y actualizar la pantalla y para capturar la información que se pasa al Applet desde el archivo html a través del tag Applet.

- init()
- destroy()
- start()
- stop()
- resize(int width, int height)
- paint(Graphics g)
- update(Graphics g)
- repaint()
- getParameter(String attr)
- getDocumentBase()
- getCodeBase()
- print(Graphics g)

4.8.3.- Un applet básico en Java

Java utiliza la extensión *.Java* para designar los ficheros fuente.
Ejemplo: código fuente del applet HolaMundo
Designamos el código en un fichero fuente Java como HolaMundo.Java.

```
//
// Applet HolaMundo de ejemplo
//
import Java.awt.Graphics;
import Java.applet.Applet;

public class HolaMundo extends Applet {
   public void paint( Graphics g ) {
      g.drawString( "Hola Mundo!",25,25 ) ;
      }
   }
```

4.9.- Exepciones

Las Excepciones en Java sirven para la detección y corrección de errores. Si hay un error, la aplicación no debería morirse sino que se debería lanzar una excepción que nosotros deberíamos capturar para resolver la situación de error. Aumentan en gran medida la robustez de las aplicaciones, siendo utilizadas en forma adecuada. Cuando se produce un error se debería lanzar una excepción indicándola expresamente. Se pueden definir excepciones extendiendo la clase *Exception*. También pueden producirse excepciones en forma implícita cuando se realiza alguna acción ilegal o no válida.

Existen exepciones predefinidas que se conocen como *exepciones runtime* de las cuales detallamos las más frecuentes:

- AritmeticException
  Surgen como el resultado de una división por cero.
- NullPointerException
  Se produce cuando se intenta acceder a una variable o método antes de haberlo definido.
- ClassCastException
  Se produce cuando se quiere convertir un objeto a una clase que no es válida.
- NegativeArraySizeException
  Surge si hay un error aritmético al intentar cambiar el tamaño de un vector.
- OutOfMemoryException
  Se produce si se intenta crear un objeto con el operador New y no hay memoria suficiente.
- NoClassDefFoundException
  Se produce si se realiza una referencia a una clase que el sistema no encuentra.
- ArrayIndexOutOfBoundsException
  Se genera al intentar acceder a un elemento de un array que exede los límites definidos inicialmente para el mismo.
- InternalException
  Este error se reserva para eventos que no deberían ocurrir.

Se pueden crear excepciones propias extendiendo la clase *System.exception*. Cualquier método que lance una excepción deberá capturarla o declararla como parte de la interface del método.

- Try
  Es el bloque de código que prevee se genere una excepción, deberá ir seguido por una cláusula catch o finally. No hay ninguna sobrecarga al sistema por incorporar sentencias try al código. La sobrecarga se produce cuando se genera la excepción.
- Catch
  Es el código que se ejecuta cuando se produce la excepción. En este bloque no puede haber código que genere excepciones. Comprueba los argumentos en el mismo orden en que aparecen en el programa, si alguno coincide se ejecuta el bloque y sigue el flujo de control por el bloque finally y concluye el control de la excepción.
- Finally
  Es el bloque de código que se ejecuta siempre haya o no una excepción. Es un trozo de código que se ejecuta independientemente de lo que se realice en el bloque try.

Cuando tratamos una excepción se nos plantea el problema qué acciones vamos a tomar y en la mayoría de los casos bastará con presentar al usuario un mensaje de error para que él decida si desea o no seguir con la ejecución del programa.

Si una excepción no es tratada en la rutina donde se produce, esta se propaga hacia arriba, buscando un bloque *try...catch* más allá de la llamada pero dentro del método que lo invocó. Si la excepción se propaga hasta lo alto de la pila de llamadas sin

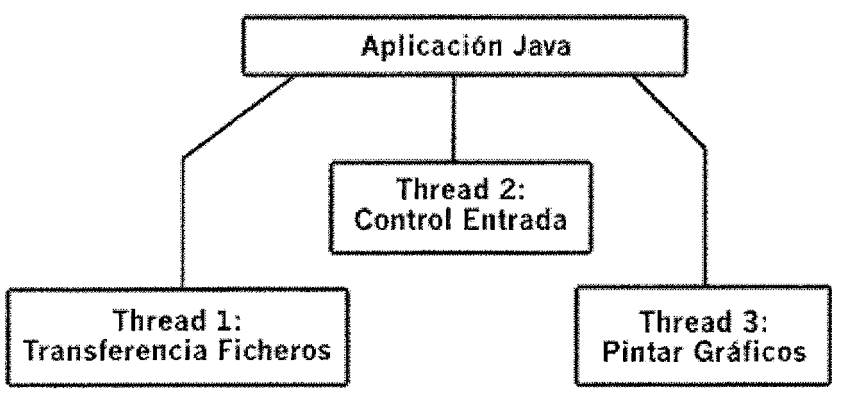
encontrar un controlador específico para ella entonces se detendrá la ejecución imprimiendo un mensaje de error.

Podemos decir que mediante la implementación de excepciones contamos con un método más seguro para el control de errores. [13]

4.10.- Threads

Java posibilita la creación y control de *threads* (*hilo*, *flujo de control del programa*) explícitamente. La simplicidad para crear, configurar y ejecutar threads, permite que se puedan implementar poderosas applets. Mientras que los programas de flujo único pueden realizar su tarea ejecutando las subtareas secuencialmente, un programa multithreaded permite que cada thread comience y termine tan pronto como sea posible. Este comportamiento presenta una mejor respuesta a la entrada en tiempo real.

Considerando el entorno multithread, cada thread representa un proceso individual ejecutándose en un sistema. A veces se les llama procesos ligeros o contextos de ejecución. Típicamente, cada thread controla un único aspecto dentro de un programa, como puede ser supervisar la entrada en un determinado periférico o controlar toda la entrada/salida del disco. Todos los threads comparten los mismos recursos, al contrario que los procesos en donde cada uno tiene su propia copia de código y datos (separados unos de otros). Gráficamente, los threads se parecen en su funcionamiento a lo que muestra la figura siguiente:



- Programas de flujo único

  Un programa de flujo único o mono-hilvanado (single-thread) utiliza un único flujo de control (thread) para controlar su ejecución. Muchos programas no necesitan la potencia o utilidad de múltiples flujos de control.

- Programas de flujo múltiple

  La utilización de threads en Java, permite una enorme flexibilidad a los programadores a la hora de plantearse el desarrollo de aplicaciones. La simplicidad para crear, configurar y ejecutar threads, permite que se puedan implementar muy poderosas y portables aplicaciones/applets que no se puede con otros lenguajes de tercera generación. En un lenguaje orientado a Internet como es Java, esta herramienta es vital.

Si se ha utilizado un navegador con soporte Java, ya se habrá visto el uso de múltiples threads en Java. Habrá observado que dos applet se pueden ejecutar al mismo tiempo, o que puede desplazar la página del navegador mientras el applet continúa ejecutándose. Esto no significa que el applet utilice múltiples threads, sino que el navegador es multithreaded.
Las aplicaciones (y applets) multithreaded utilizan muchos contextos de ejecución para cumplir su trabajo. Hacen uso del hecho de que muchas tareas contienen subtareas distintas e independientes. Se puede utilizar un thread para cada sub tarea. Mientras que los programas de flujo único pueden realizar su tarea ejecutando las sub tareas secuencialmente, un programa multithreaded permite que cada thread comience y termine tan pronto como sea posible. Este comportamiento presenta una mejor respuesta a la entrada en tiempo real.

4.10.1.- Creación de un Thread

Hay dos formas de obtener threads en Java:

- Implementando la interface Runnable , la forma habitual de crear threads. La interface define el trabajo, y las clases o clase que implementan la interface realizan ese trabajo
- Extendiendo la clase Thread se pueden heredar los métodos y variables de la clase padre.

La implementación de la interface Runnable es la forma habitual de crear threads. Las interfaces proporcionan al programador una forma de agrupar el trabajo de infraestructura de una clase. Se utilizan para diseñar los requerimientos comunes al conjunto de clases a implementar. La interface define el trabajo y la clase, o clases, que implementan la interface realizan ese trabajo. Los diferentes grupos de clases que implementen la interface tendrán que seguir las mismas reglas de funcionamiento.

Hay una cuantas diferencias entre interface y clase. Primero, una interface solamente puede contener métodos abstractos y/o variables estáticas y finales (constantes). Las clases, por otro lado, pueden implementar métodos y contener variables que no sean constantes. Segundo, una interface no puede implementar cualquier método. Una clase que implemente una interface debe implementar todos los métodos definidos en esa interface. Una interface tiene la posibilidad de poder extenderse de otras interfaces y, al contrario que las clases, puede extenderse de múltiples interfaces. Además, una interface no puede ser instanciada con el operador new; por ejemplo, la siguiente sentencia no está permitida:

```
Runnable a = new Runnable();  // No se permite
```

El primer método de crear un thread es simplemente extender la clase Thread:

```
class MiThread extends Thread {
  public void run() {
    . . .
    }
```

4.10.2.- Arranque de un Thread

Las aplicaciones ejecutan main() tras arrancar. Esta es la razón de que main() sea el lugar natural para crear y arrancar otros threads. El código para crear un nuevo thread es:

```
t1 = new TestTh( "Thread 1",(int)(Math.random()*2000) );
```

Los dos argumentos pasados representan el nombre del thread y el tiempo que queremos que espere antes de imprimir el mensaje.

Al tener control directo sobre los threads, tenemos que arrancarlos explícitamente con:

```
t1.start();
```

start(), en realidad es un método oculto en el thread que llama al método run().

4.10.3.- Manipulación de un Thread

Si todo fue bien en la creación del thread, t1 debería contener un thread válido, que controlaremos en el método run().

Una vez dentro de run(), podemos comenzar las sentencias de ejecución como en otros programas. run() sirve como rutina main() para los threads; cuando run() termina, también lo hace el thread. Todo lo que queramos que haga el thread ha de estar dentro de run(), por eso cuando decimos que un método es Runnable, nos obliga a escribir un método run().

Cuando se quiera retrasar la ejecucuón de un theard se deberá utilizar la instrucción:

```
sleep( retardo );
```

No consume recursos del sistema mientras el thread duerme. De esta forma otros threads pueden seguir funcionando.

4.10.4.- Suspensión de un Thread

Puede resultar útil suspender la ejecución de un thread sin marcar un límite de tiempo. Si, por ejemplo, está construyendo un applet con un thread de animación, querrá permitir al usuario la opción de detener la animación hasta que quiera continuar. No se trata de terminar la animación, sino desactivarla. Para este tipo de control de thread se puede utilizar el método suspend().

```
t1.suspend();
```

Este método no detiene la ejecución permanentemente. El thread es suspendido indefinidamente y para volver a activarlo de nuevo necesitamos realizar una invocación al método resume():

```
t1.resume();
```

4.10.5.- Parada de un Thread

El último elemento de control que se necesita sobre threads es el método stop(). Se utiliza para terminar la ejecución de un thread:

```
t1.stop();
```

Esta llamada no destruye el thread, sino que detiene su ejecución. La ejecución no se puede reanudar ya con t1.start().

Cuando se desasignen las variables que se usan en el thread, el objeto thread (creado con new) quedará marcado para eliminarlo y el garbage collector se encargará de liberar la memoria que utilizaba.

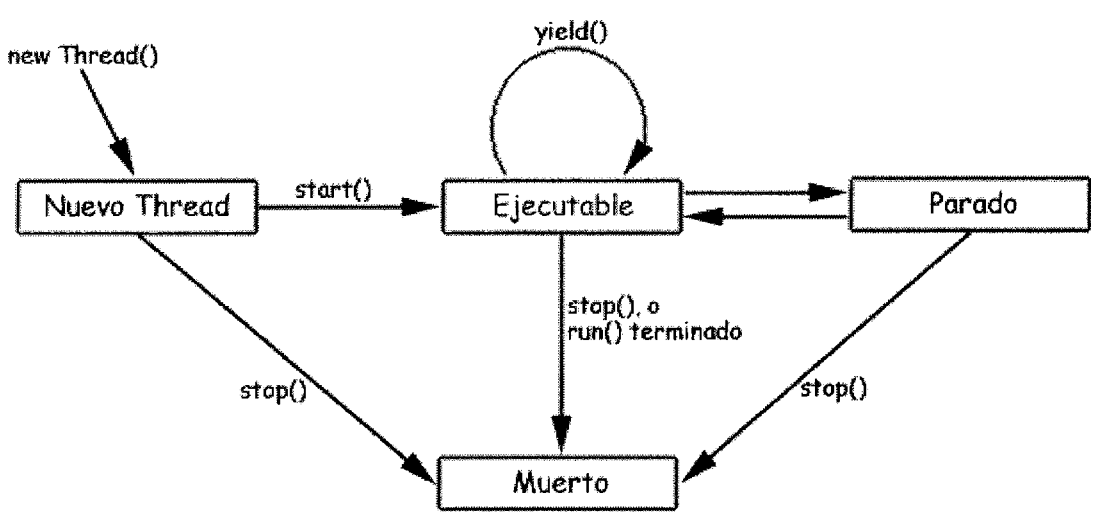
Si se necesita, se puede comprobar si un thread está vivo o no; considerando vivo un thread que ha comenzado y no ha sido detenido.

```
t1.isAlive();
```

Este método devolverá true en caso de que el thread t1 esté vivo, es decir, ya se haya llamado a su método run() y no haya sido parado con un stop() ni haya terminado el método run() en su ejecución.

4.10.6.- Estados de un Thread

Durante el ciclo de vida de un thread, éste se puede encontrar en diferentes estados. La figura siguiente muestra estos estados y los métodos que provocan el paso de un estado a otro. Este diagrama no es una máquina de estados finita, pero es lo que más se aproxima al funcionamiento real de un thread.



La siguiente sentencia crea un nuevo thread pero no lo arranca, lo deja en el estado de "Nuevo Thread":

```
Thread MiThread = new MiClaseThread();
```

Cuando un thread está en este estado, es simplemente un objeto Thread vacío. El sistema no ha destinado ningún recurso para él. Desde este estado solamente puede arrancarse llamando al método start(), o detenerse definitivamente, llamando al método stop(); la llamada a cualquier otro método carece de sentido y lo único que provocará será la generación de una excepción.

- Estado ejecutable

```
Thread MiThread = new MiClaseThread();
MiThread.start();
```

  La llamada al método start() creará los recursos del sistema necesarios para que el thread puede ejecutarse, lo incorpora a la lista de procesos disponibles para ejecución del sistema y llama al método run() del thread. En este momento nos encontramos en el estado "Ejecutable" del diagrama. Y este estado es Ejecutable y no En Ejecución, porque cuando el thread está aquí no esta corriendo. Muchos ordenadores tienen solamente un procesador lo que hace imposible que todos los threads estén corriendo al mismo tiempo. Java implementa un tipo de scheduling o lista de procesos, que permite que el procesador sea compartido entre todos los procesos o threads que se encuentran en la lista. Sin embargo, para nuestros propósitos, y en la mayoría de los casos, se puede considerar que este estado es realmente un estado "En Ejecución", porque la impresión que produce ante nosotros es que todos los procesos se ejecutan al mismo tiempo.

  Cuando el thread se encuentra en este estado, todas las instrucciones de código que se encuentren dentro del bloque declarado para el método run(), se ejecutarán secuencialmente.

- Estado Parado

  El thread entra en estado "Parado" cuando alguien llama al método suspend(), cuando se llama al método sleep(), cuando el thread está bloqueado en un proceso de entrada/salida o cuando el thread utiliza su método wait() para esperar a que se cumpla una determinada condición. Cuando ocurra cualquiera de las cuatro cosas anteriores, el thread estará Parado.

  Para cada una de los cuatro modos de entrada en estado Parado, hay una forma específica de volver a estado Ejecutable. Cada forma de recuperar ese estado es exclusiva; por ejemplo, si el thread ha sido puesto a dormir, una vez transcurridos los milisegundos que se especifiquen, él solo se despierta y vuelve a estar en estado Ejecutable. Llamar al método resume() mientras esté el thread durmiendo no serviría para nada.

  Los métodos de recuperación del estado Ejecutable, en función de la forma de llegar al estado Parado del thread, son los siguientes:

  - Si un thread está dormido, pasado el lapso de tiempo
  - Si un thread está suspendido, luego de una llamada al método resume()
  - Si un thread está bloqueado en una entrada/salida, una vez que el comando E/S concluya su ejecución

- Si un thread está esperando por una condición, cada vez que la variable que controla esa condición varíe debe llamarse a notify() o notifyAll()

- Estado Muerto

  Un thread se puede morir de dos formas: por causas naturales cuando concluye de forma habitual su método run() o invocando a su método stop().

  Los applets utilizarán el método stop() para matar a todos sus threads cuando el navegador con soporte Java en el que se están ejecutando le indica al applet que se detengan, por ejemplo, cuando se minimiza la ventana del navegador o cuando se cambia de página. [13]

4.11.- Comunicaciones

El sistema de Entrada/Salida de Unix sigue el paradigma que normalmente se designa como Abrir-Leer-Escribir-Cerrar. Antes de que un proceso de usuario pueda realizar operaciones de entrada/salida, debe hacer una llamada a Abrir (open) para indicar, y obtener permisos para su uso, el fichero o dispositivo que quiere utilizar. Una vez que el objeto está abierto, el proceso de usuario realiza una o varias llamadas a Leer (read) y Escribir (write), para conseguir leer y escribir datos. Leer toma datos desde el objeto y los transfiere al proceso de usuario, mientras que Escribir transfiere datos desde el proceso de usuario al objeto. Una vez que todos estos intercambios de información estén concluidos, el proceso de usuario llamará a Cerrar (close) para informar al sistema operativo que ha finalizado la utilización del objeto que antes había abierto.

Cuando se incorporan las características a Unix de comunicación entre procesos (IPC) y el manejo de redes, la idea fue implementar la interface con IPC similar a la que se estaba utilizando para la entrada/salida de ficheros, es decir, siguiendo el paradigma del párrafo anterior. En Unix, un proceso tiene un conjunto de descriptores de entrada/salida desde donde Leer y por donde Escribir. Estos descriptores pueden estar referidos a ficheros, dispositivos, o canales de comunicaciones (sockets). El ciclo de vida de un descriptor, aplicado a un canal de comunicación (socket), está determinado por tres fases (siguiendo el paradigma):

Creación, apertura del socket
Lectura y Escritura, recepción y envío de datos por el socket
Destrucción, cierre del socket

La interface IPC en Unix-BSD está implementada sobre los protocolos de red TCP y UDP. Los destinatarios de los mensajes se especifican como direcciones de socket; cada dirección de socket es un identificador de comunicación que consiste en una dirección Internet y un número de puerto.

Las operaciones IPC se basan en pares de sockets. Se intercambian información transmitiendo datos a través de mensajes que circulan entre un socket en un proceso y otro socket en otro proceso. Cuando los mensajes son enviados, se encolan en el socket hasta que el protocolo de red los haya transmitido. Cuando llegan, los mensajes son encolados en el socket de recepción hasta que el proceso que tiene que recibirlos haga las llamadas necesarias para recoger esos datos.

4.12.- Sockets

Los Sockets son puntos finales de enlaces de comunicaciones entre procesos. Los procesos los tratan como descriptores de ficheros, de forma que se pueden intercambiar datos con otros procesos transmitiendo y recibiendo a través de sockets.

El tipo de sockets describe la forma en la que se transfiere información a través de ese socket.

- *Sockets Stream (TCP, Transmission Control Protocol)*

  Son un servicio orientado a conexión donde los datos se transfieren sin encuadrarlos en registros o bloques. Si se rompe la conexión entre los procesos, éstos serán informados.

  El protocolo de comunicaciones con streams es un protocolo orientado a conexión, ya que para establecer una comunicación utilizando el protocolo TCP, hay que establecer en primer lugar una conexión entre un par de sockets.

  Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita una conexión (cliente). Una vez que los dos sockets estén conectados, se pueden utilizar para transmitir datos en ambas direcciones.

- *Sockets Datagrama (UDP, User Datagram Protocol)*

  Son un servicio de transporte sin conexión. Son más eficientes que TCP, pero no está garantizada la fiabilidad. Los datos se envían y reciben en paquetes, cuya entrega no está garantizada. Los paquetes pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.

  El protocolo de comunicaciones con datagramas es un protocolo sin conexión, es decir, cada vez que se envíen datagramas es necesario enviar el descriptor del socket local y la dirección del socket que debe recibir el datagrama. Como se puede ver, hay que enviar datos adicionales cada vez que se realice una comunicación.

- *Sockets Raw*

  Son sockets que dan acceso directo a la capa de software de red subyacente o a protocolos de más bajo nivel. Se utilizan sobre todo para la depuración del código de los protocolos.

4.12.1.- Diferencias entre Sockets Stream y Datagrama

Ahora se nos presenta un problema, ¿qué protocolo, o tipo de sockets, debemos usar - UDP o TCP? La decisión depende de la aplicación cliente/servidor que estemos escribiendo. Vamos a ver algunas diferencias entre los protocolos para ayudar en la decisión.

En UDP, cada vez que se envía un datagrama, hay que enviar también el descriptor del socket local y la dirección del socket que va a recibir el datagrama, luego éstos son más grandes que los TCP. Como el protocolo TCP está orientado a conexión, tenemos que establecer esta conexión entre los dos sockets antes de nada, lo que implica un cierto tiempo empleado en el establecimiento de la conexión, que no existe en UDP.

En UDP hay un límite de tamaño de los datagramas, establecido en 64 kilobytes, que se pueden enviar a una localización determinada, mientras que TCP no tiene límite; una vez que se ha establecido la conexión, el par de sockets funciona como los streams: todos los datos se leen inmediatamente, en el mismo orden en que se van recibiendo.

UDP es un protocolo desordenado, no garantiza que los datagramas que se hayan enviado sean recibidos en el mismo orden por el socket de recepción. Al contrario, TCP es un protocolo ordenado, garantiza que todos los paquetes que se envíen serán recibidos en el socket destino en el mismo orden en que se han enviado.

Los datagramas son bloques de información del tipo lanzar y olvidar. Para la mayoría de los programas que utilicen la red, el usar un flujo TCP en vez de un datagrama UDP es más sencillo y hay menos posibilidades de tener problemas. Sin embargo, cuando se requiere un rendimiento óptimo, y está justificado el tiempo adicional que supone realizar la verificación de los datos, los datagramas son un mecanismo realmente útil.

En resumen, TCP parece más indicado para la implementación de servicios de red como un control remoto (rlogin, telnet) y transmisión de ficheros (ftp); que necesitan transmitir datos de longitud indefinida. UDP es menos complejo y tiene una menor sobrecarga sobre la conexión; esto hace que sea el indicado en la implementación de aplicaciones cliente/servidor en sistemas distribuidos montados sobre redes de área local.

4.12.2.- Uso de sockets

Podemos pensar que un Servidor Internet es un conjunto de sockets que proporciona capacidades adicionales del sistema, los llamados servicios.

Puertos y Servicios

Cada servicio está asociado a un puerto. Un puerto es una dirección numérica a través de la cual se procesa el servicio.

Sobre un sistema Unix, los servicios que proporciona ese sistema se indican en el fichero /etc/services, y algunos ejemplos son:

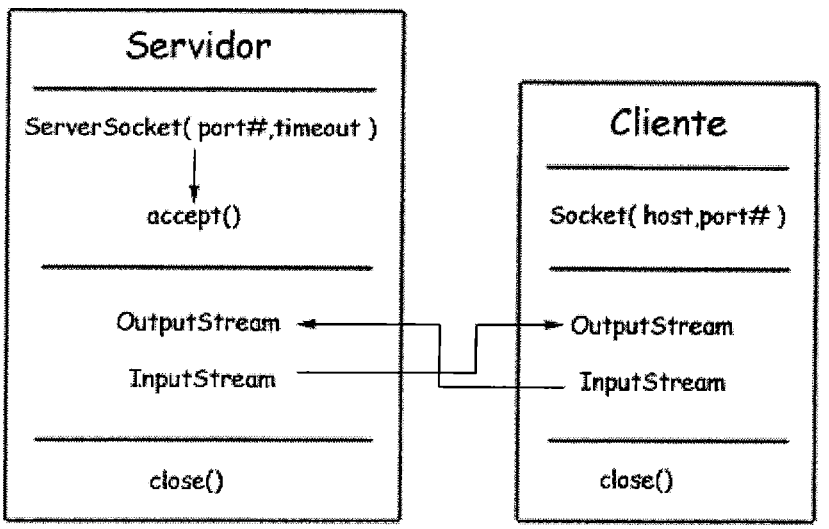
```
daytime   13/udp
ftp       21/tcp
telnet    23/tcp   telnet
smtp      25/tcp   mail
http      80/tcp
```

La primera columna indica el nombre del servicio. La segunda columna indica el puerto y el protocolo que está asociado al servicio. La tercera columna es un alias del servicio; por ejemplo, el servicio smtp, también conocido como mail, es la implementación del servicio de correo electrónico.

Las comunicaciones de información relacionada con Web tienen lugar a través del puerto 80 mediante protocolo TCP. Para emular esto en Java, usaremos la clase Socket. La fecha (daytime). Sin embargo, el servicio que toma la fecha y la hora del sistema, está ligado al puerto 13 utilizando el protocolo UDP. Un servidor que lo emule en Java usaría un objeto DatagramSocket.

4.12.3.- Modelo de comunicaciones con Java

En Java, crear una conexión socket TCP/IP se realiza directamente con el paquete Java.net. A continuación mostramos un diagrama de lo que ocurre en el lado del cliente y del servidor:



El modelo de sockets más simple es:

El servidor establece un puerto y espera durante un cierto tiempo (timeout segundos), a que el cliente establezca la conexión. Cuando el cliente solicite una conexión, el servidor abrirá la conexión socket con el método accept().

El cliente establece una conexión con la máquina host a través del puerto que se designe en puerto#

El cliente y el servidor se comunican con manejadores InputStream y OutputStream

Hay una cuestión al respecto de los sockets, que viene impuesta por la implementación del sistema de seguridad de Java.

Actualmente, los applets sólo pueden establecer conexiones con el nodo desde el cual se transfirió su código. Esto está implementado en el JDK y en el intérprete de Java de Netscape. Esto reduce en gran manera la flexibilidad de las fuentes de datos disponibles para los applets. El problema si se permite que un applet se conecte a cualquier máquina de la red, es que entonces se podrían utilizar los applets para inundar la red desde un ordenador con un cliente Netscape del que no se sospecha y sin ninguna posibilidad de rastreo.

4.12.4.- Apertura de Sockets

Si estamos programando un cliente, el socket se abre de la forma:

```
Socket miCliente;
miCliente = new Socket( "maquina",numeroPuerto );
```

Donde maquina es el nombre de la máquina en donde estamos intentando abrir la conexión y número Puerto es el puerto (un número) del servidor que está corriendo sobre el cual nos queremos conectar. Cuando se selecciona un número de puerto, se debe tener en cuenta que los puertos en el rango 0-1023 están reservados para usuarios con muchos privilegios (super usuarios o root). Estos puertos son los que utilizan los

servicios estándar del sistema como email, ftp o http. Para las aplicaciones que se desarrollen, asegurarse de seleccionar un puerto por encima del 1023.

En el ejemplo anterior no se usan excepciones; sin embargo, es una gran idea la captura de excepciones cuando se está trabajando con sockets. El mismo ejemplo quedaría como:

```
Socket miCliente;
try {
   miCliente = new Socket( "maquina",numeroPuerto );
} catch( IOException e ) {
   System.out.println( e );
   }
```

Si estamos programando un servidor, la forma de apertura del socket es la que muestra el siguiente ejemplo:

```
Socket miServicio;
try {
   miServicio = new ServerSocket( numeroPuerto );
} catch( IOException e ) {
   System.out.println( e );
   }
```

A la hora de la implementación de un servidor también necesitamos crear un objeto socket desde el ServerSocket para que esté atento a las conexiones que le puedan realizar clientes potenciales y poder aceptar esas conexiones:

```
Socket socketServicio = null;
try {
   socketServicio = miServicio.accept();
} catch( IOException e ) {
   System.out.println( e );
   }
```

4.12.5.- Cierre de Sockets

Siempre deberemos cerrar los canales de entrada y salida que se hayan abierto durante la ejecución de la aplicación. En la parte del cliente:

```
try {
   salida.close();
   entrada.close();
   miCliente.close();
} catch( IOException e ) {
   System.out.println( e );
   }
```

Y en la parte del servidor:

```
try {
   salida.close();
   entrada.close();
   socketServicio.close();
   miServicio.close();
} catch( IOException e ) {
   System.out.println( e );
   }
```

Capítulo 5

JDBC (Java Database Conectivity)

5.1.- Qué es JDBC ?

JDBC consiste en un conjunto de clases e interfaces escritos en el lenguaje de programación Java. Es un paquete *java.sql* que provee un sencillo mecanismo para enviar consultas SQL a una Base de Datos y recibir el resultado de dichas consultas.

Se debe notar que JDBC es una API para SQL, no un mecanismo para embeber sentencias SQL en programas Java.

Usando JDBC, es fácil enviar sentencias SQL virtualmente a cualquier Base de Datos relacional. En otras palabras, con el JDBC API, no es necesario escribir un programa para acceder a la Base de Datos Sybase, otro programa para acceder a la base Oracle, otro para Informix, y así siguiendo. Solo podemos escribir un simple programa usando JDBC API, y el programa estaría habilitado para enviar sentencias SQL para la Base de Datos apropiada. Y con una aplicación escrita en lenguaje Java, tampoco debe preocuparse en escribir distintas aplicaciones para correr sobre distintas plataformas. La combinación de JAVA y JDBC le provee al programador escribir el programa una sola vez y correrlo sobre cualquier plataforma.

5.2.- Clases de JDBC

JDBC provee tres tipos de clases:

- Para la conexión
- Para el procesamiento
- Para el soporte

- Clases para la conexión

java.sql.DriverManager
: Controla los drivers que son registrados por él

java.sql.Driver
: Maneja un tipo específico de Base de Datos

java.sql.DriverPropertyInfo
: Configura y retorna propiedades de la conexión

java.sql.Connection
: Es una conexión con una Base de Datos específica. A través de una conexión individual las sentencias SQL son enviadas a la base, son ejecutadas, y los resultados son retornados.

- Clases para el procesamiento

java.sql.DatabaseMetaData
: Devuelve información de la base

java.sql.Statement

Envía una sentencia SQL a la base

java.sql.CallableStatement

Similar a Statement, para ejecutar store procedures

java.sql.PreparedStatement

Similar a Statement, pero para sentencias precompiladas

java.sql.ResultSet

Es el resultado de la ejecución de una sentencia SQL

java.sql.ResultSetMetaData

Devuelve información sobre los tipos y propiedades en ResultSet

- Clases para el Soporte

java.sql.Types

Constantes usadas en SQL

java.sql.Numeric

Soporte para números de precisión arbitraria

java.sql.Date

Soporte para fechas

java.sql.Time

Soporte para hora

java.sql.Timestamp

Soporte para fecha y hora combinados

java.sql.SQLException

Clase que contiene información acerca de un error ocurrido

java.sql.SQLWarnings

subclase de SQLException

5.3.- Qué se puede hacer con JDBC?

Por ejemplo, con Java y JDBC API , es posible publicar una página Web que contenga un applet que usa información obtenida de una Base de Datos remota. O una empresa puede usar JDBC para conectar a todos sus empleados a una o más Base de Datos internas vía una Intranet.

Con más y más programadores usando el lenguaje de programación Java, la necesidad de acceder fácilmente a la Base de Datos desde Java crece continuamente.

5.4.- Qué hace JDBC?

- Establece una conexión con la Base de Datos.
- Envía sentencias SQL
- Procesa resultados

El siguiente fragmento de código es un ejemplo básico de los tres pasos mencionados:

```
Connection con = DriverManager.getConnection(
                    "jdbc:odbc:wombat", "login", "password");
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery("SELECT a,b,c FROM table1");
```

```
while (rs.next()) {
        int x = getInt("a");
        string  s = getString("b");
        float f = getFloat("c");
}
```

JDBC es una interface a *bajo nivel* , el cual significa que es usada para invocar directamente comandos SQL. Trabaja muy bien en esta capacidad y es más fácil de usar que otras APIs de conexiones a Base de Datos, pero fue diseñada también para ser una base sobre la cual construir herramientas e interfaces de más alto nivel.

Por lo tanto , los pasos para usar JDBC son:

- Registrar e instanciar el driver (en general no se registra el driver, sólo es necesaria si se trabaja con mas de una)
- Crear la conexión usando la clase Connection
- Usar métodos de la API para interactuar con la fuente de datos usando la clase Statement
- Obtener los resultados usando métodos de la API en la clase ResultSet

5.5.- JDBC versus ODBC

El API ODBC de Microsoft (Open Database Conectivity) es probablemente la interface de programación ampliamente más usada para acceder a la Base de Datos relacional. Ofrece la posibilidad para conectarse a casi todas las Bases de Datos sobre casi todas las plataformas.

La pregunta ahora es, porqué no usar ODBC desde Java?

La respuesta es usar ODBC para Java, pero es mejor con la ayuda de JDBC, como un puente JDBC-ODBC.

Porqué necesitamos JDBC?

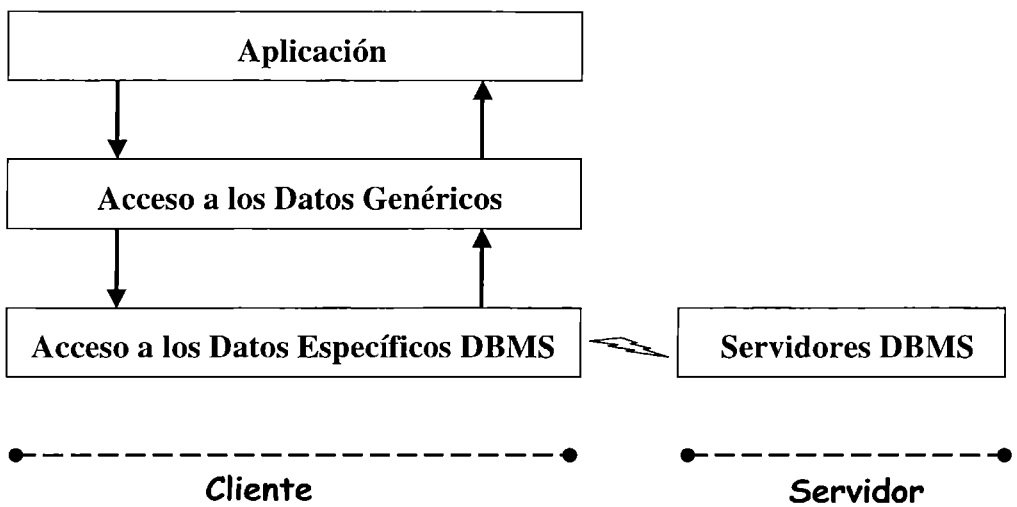
- ODBC no es apropiado para usar directamente desde Java porque usa interface C.
- Una traducción literal del ODBC C API al Java API no sería aconsejable. Por ejemplo, Java no tiene punteros, y ODBC hace un uso intensivo de ellos, incluyendo el puntero genérico que notoriamente tiende al error llamado *'void'*. Uno puede pensar en JDBC como ODBC traducido a una interface orientada a objetos que sea natural para los programadores de Java.
- ODBC es difícil de aprender. Mezcla características simples y avanzadas juntas, y tiene opciones complejas aun para consultas simples. JDBC, por otro lado, fue diseñado para mantener cosas simples de manera simple mientras que permite más capacidades avanzadas donde sean requeridas.
- Un Java API como JDBC es necesario para permitir una solución Java pura.. Cuando se usa ODBC, el manejador de driver ODBC y los drivers deben ser manualmente instalados en toda máquina cliente. Cuando el driver JDBC es escrito completamente en Java, sin embargo, el código JDBC es automáticamente instalable, portable, y seguro en todas las plataformas Java desde computadoras en red a mainframes.

En resumen, el JDBC API es una interface Java natural para los conceptos y abstracciones SQL básicos. Construye sobre ODBC en vez de comenzar de cero, entonces los programadores familiarizados con el ODBC encontraran muy fácil aprender JDBC.

JDBC retiene las características del diseño básico de ODBC; de hecho, ambas interfaces están basadas en X/Open SQL CLI (Call Level Interface). La gran diferencia es que JDBC construye sobre y refuerza el estilo y las virtudes de Java, y por supuesto, es fácil de usar.

5.6.- Acceso de datos a través de JDBC

En la arquitectura two-tier, la aplicación cliente usa la capa de acceso de datos para comunicarse con el servidor. La comunicación entre el cliente y el servidor se hace a través del protocolo de red. El servidor reside en un sistema distinto al del cliente



Data Access Layer Revised

5.7.- Drivers JDBC

JavaSoft implementó la capa de acceso de datos en Microsoft's ODBC API, el cual es, probablemente, la más ampliamente usada para acceder a la Base de Datos Relacional en sistemas operativos Windows.

5.7.1.- Categorías de Drivers

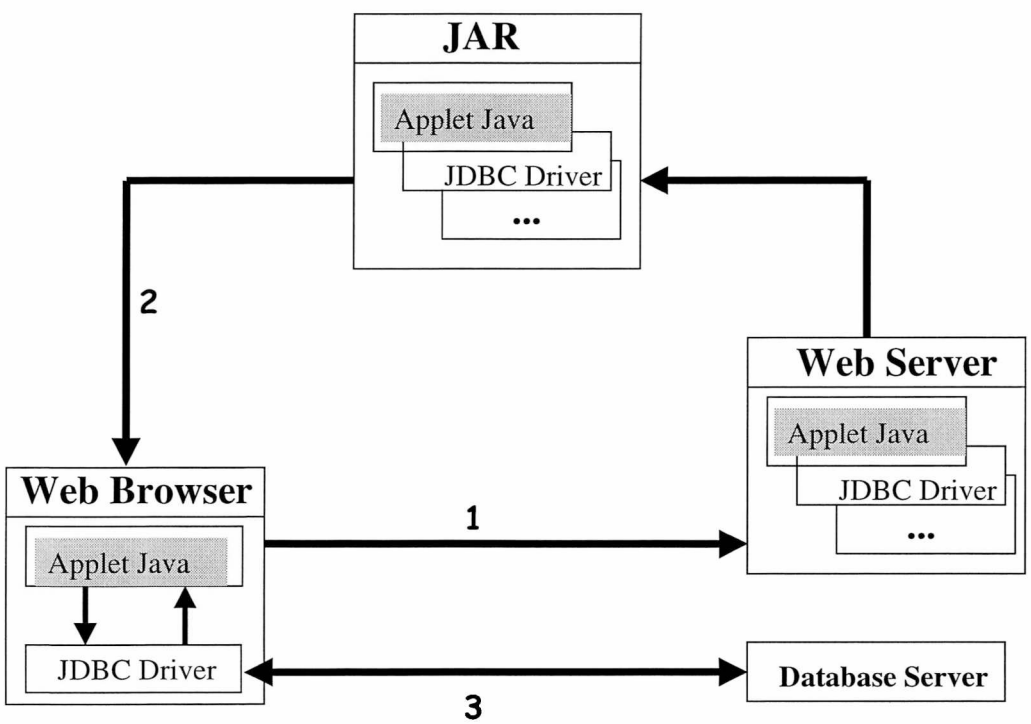
- Categoría 1
  Bridge JDBC-ODBC: provee acceso JDBC a través de los drivers ODBC. En este caso se deberán instalar los drivers apropiados en cada máquina cliente
- Categoría 2

Native API partly-Java driver: este tipo de driver convierte las llamadas JDBC en llamadas al API cliente para Oracle, Sybase, Informix u otros DBMS. Nótese que, como el driver bridge, requiere que algún código binario sea cargado en cada máquina cliente.

- Categoría 3
  JDBC-Net pure Java driver: este driver traduce la llamada JDBC en un protoclo de red de DBMS independiente, el cual es traducido luego a un protocolo DBMS por el servidor. Este servidor de red intermediario puede conectar los clientes Java puro a las distintas bases de datos. En orden para que estos productos también soporten accesos a Internet deben manejar los requerimientos adicionales la Web impone para seguridad, accesos a través de firewall, y otros.
- Categoría 4
  Native-Protocol pure Java driver: convierte directamente la llamada JDBC en el protocolo de red usado por el DBMS. Esto permite una llamada directa desde la máquina Cliente al Server DBMS.

5.7.2.- Driver JDBC Ideal

Imaginemos un ambiente Internet o Intranet



Utilizamos un Web Browser (cliente), apuntamos a un URL y pedimos ejecutar una aplicación de Base de Datos desde un Web Server (1)

El servidor responde enviando el applet pedido (the database application), incluyendo el driver JDBC y otras fuentes pedidas, empaquetando en un archivo Jar al Web Browser (2)

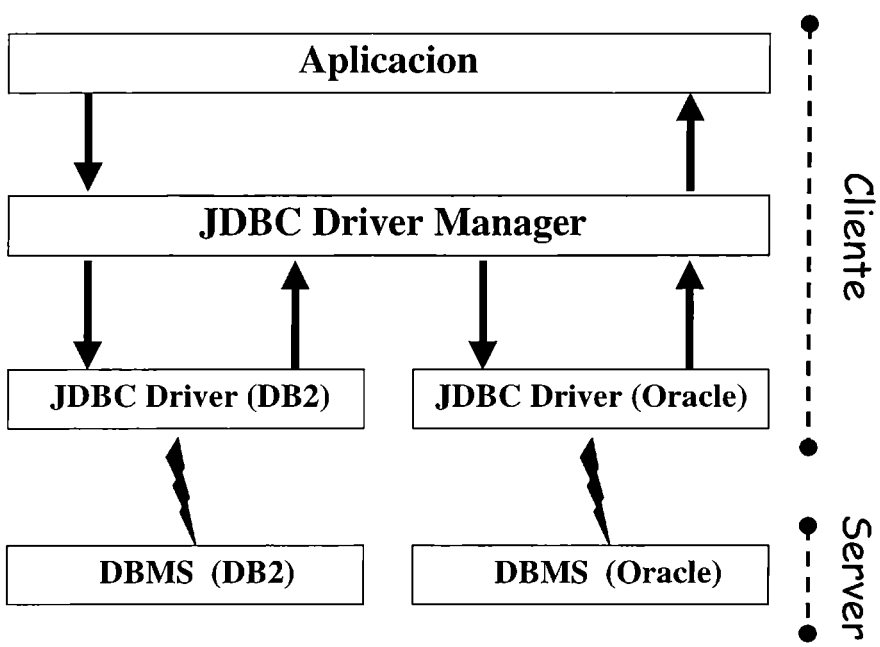
Ni bien el archivo Jar ha sido bajado exitosamente, el Web Browser comienza el applet, el cual usa el driver JDBC para acceder a la Base de Datos.

En este escenario, no se requieren programas y configuraciones adicionales en la máquina cliente, porque el módulo adicional, el driver JDBC, es automáticamente enviado con el applet en un archivo Jar . El único software requerido en el cliente es el Web Browser y tenemos una configuración *thin-client.*

Como resultado de este escenario, podemos describir los requerimientos de un driver JDBC ideal.

Un driver JDBC ideal es un driver de la Base de Datos escrito íntegramente en Java y usado por una aplicación para acceder a una Base de Datos. El servidor responde directamente a los pedidos del driver JDBC sin ninguna interface adicional.

La comunicación entre el driver JDBC y el servidor de Base de Datos es a través de un protocolo de red, el cual debe ser construido en un motor de Base de Datos. El driver JDBC habla directamente con la Base de Datos a través de *sockets.*



Un Driver JDBC Ideal

El tipo de driver es provisto en muchos casos por el proveedor de la Base de Datos. JavaSoft tiene categorizado este tipo de driver como una categoría 4

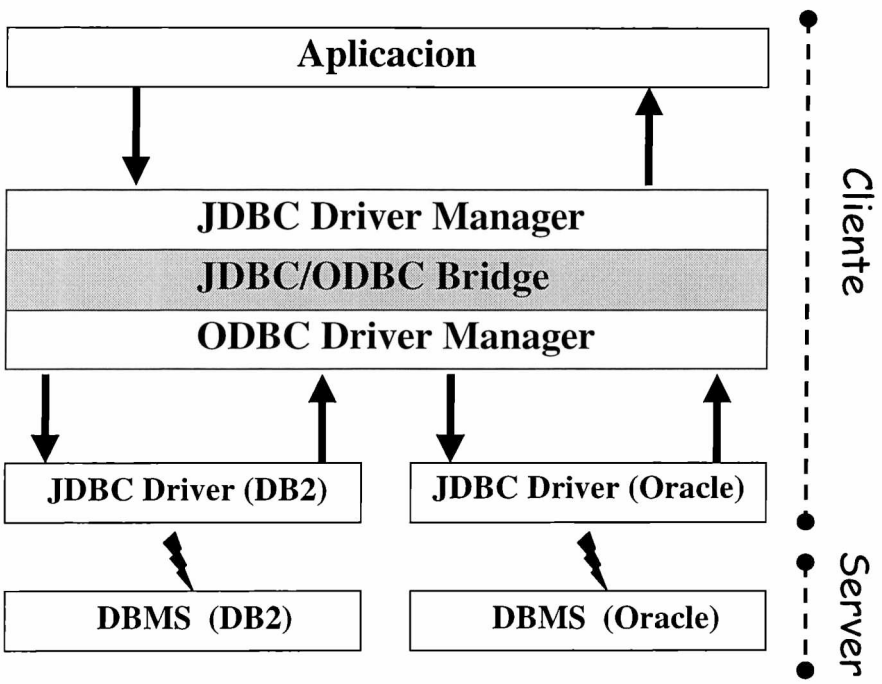
5.7.3.- Soluciones para desarrollar un Driver JDBC

JavaSoft identifica tres soluciones intermedias para desarrollar un driver JDBC para un producto existente:

- JDBC and ODBC Bridge Driver
- JDBC and Vendor Specific Bridge Driver
- JDBC Generic Network Protocol Driver

JDBC and ODBC Bridge Driver

JavaSoft provee un puente entre las aplicaciones Java y los drivers ODBC para utilizar varios drivers ODBC existentes para diferentes motores de Base de Datos



JDBC-ODBC Bridge

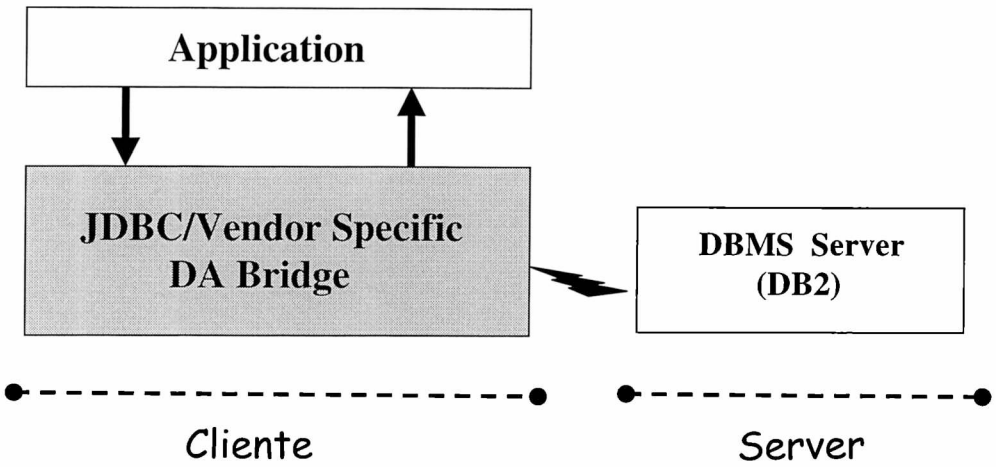
Hay una relación muy cerrada entre las arquitecturas ODBC y JDBC y APIs .

JavaSoft categoriza el driver ODBC Bridge como driver de categoría 1.

JDBC and Vendor Specific Bridge Driver
(driver puente específico entre JDBC y proveedor)

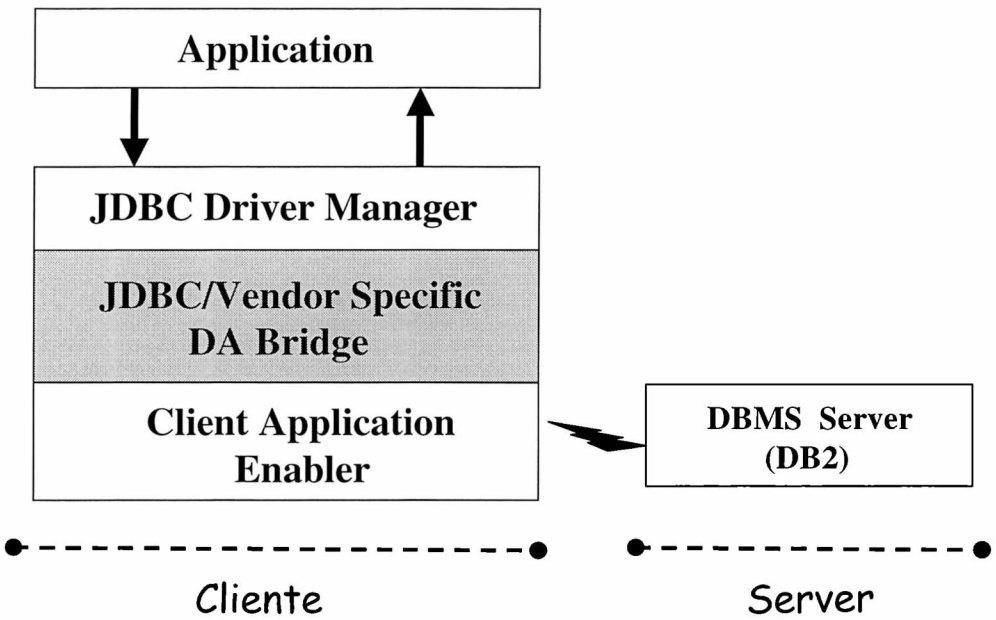
ODBC no es la única forma de acceder a la Base de Datos. Muchos cliente de Base de Datos tienen sus propios drivers para acceder a las bases.

IBM, por ejemplo usa driver DB2 Client Application Enabler (CAE) para acceder al servidor de Base de Datos DB2



Vendor Specific Driver

Para hacer uso de este tipo de driver en un ambiente Java, se requiere un JDBC/Vendor bridge.



JDBC/Vendor Specific Driver

JavaSoft categoriza este tipo de driver como categoría 2.

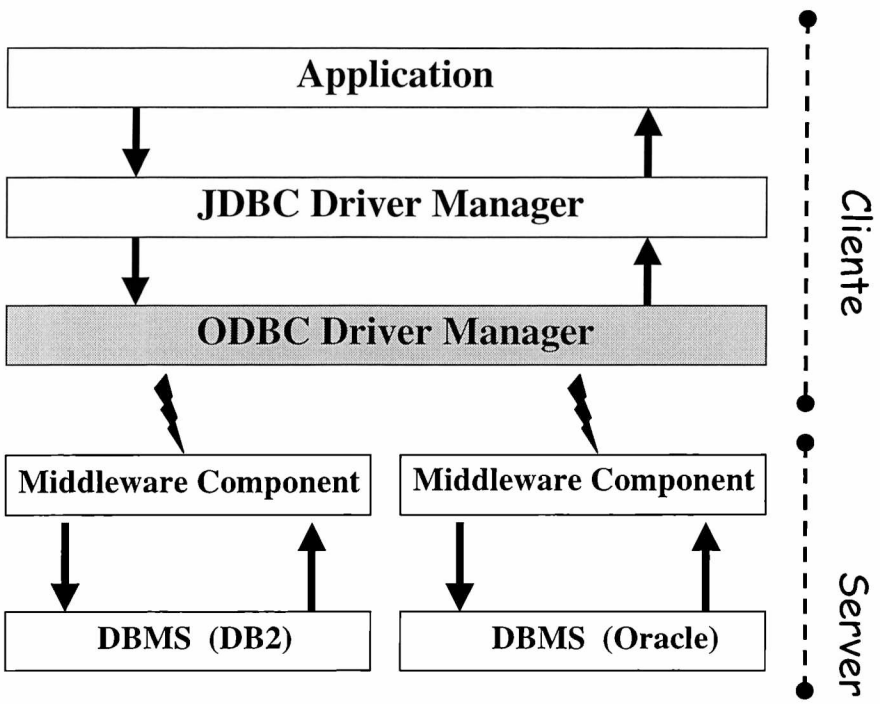
JDBC Generic Network Protocol Driver
(driver protocolo de red genérico JDBC)

Las dos soluciones intermedias introducidas no pueden ser usadas en un ambiente Internet usando driver JDBC, porque ambos manejadores de drivers depende de un conjunto de, librerías escritas en un lenguaje distinto al de Java.

Es verdadero para driver ODBC y driver específico del cliente, tal como DB2 CAE.

Para solventar este problema, dividimos el JDBC ideal en dos componentes, un componente cliente y un componente servidor, y mueve todas las funciones no escritas en lenguaje Java al componente servidor, tal que la parte del cliente puede ser escrita en lenguaje Java Puro.

La parte del cliente es responsable de traducir las llamadas de JDBC al protocolo de red de Base de Datos independiente, y el componente servidor es responsable de traducir el protocolo de red independiente de la Base de Datos en las llamadas de la Base de Datos. También se hace referencia al componente servidor como el componente *middleware*



JDBC Generic Network Protocol Driver

El driver de protocolo de red genérico JDBC es extremadamente flexible porque no requiere un código instalado sobre el cliente, y un driver simple puede proveer accesos a Base de Datos múltiples.

JavaSoft categoriza este tipo de driver como categoría 3.

Entendiendo las diferentes formas de conectarse con JDBC a la Base de Datos , podemos explicar cómo la aplicación de la Base de Datos Java es estructurada.

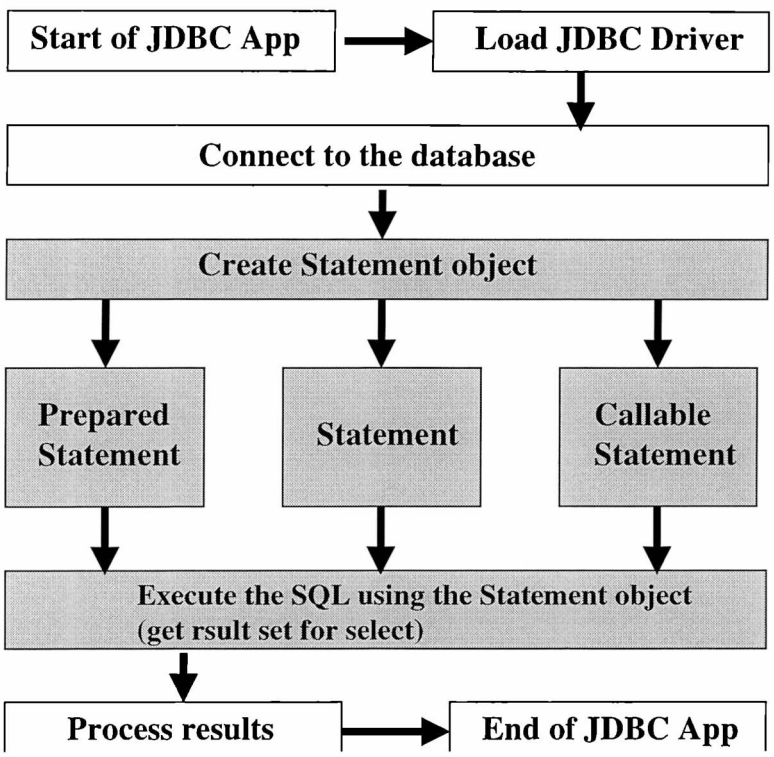
5.8.- Estructura de una Aplicación JDBC

La siguiente figura muestra la estuctura de una aplicación JDBC. Antes que podamos comunicarnos con la Base de Datos, debemos cargar el driver JDBC relacionado. Usando la clase DriverManager , cargamos el driver y luego hacemos la conexión a la base.

Una vez que la conexión es satisfactoria, creamos una instancia de la clase statement. El objeto statement es usado para representar sentencias SQL. Hay tres tipos de sentencias:

- Statement (sin variables del host)
- Prepared statement (con el lugar guardado para variables del host)
- Callable statement

El objeto *result set* maneja las filas retribuídas por un select SQL y mantiene la posición de la fila corriente. El próximo método mueve a la próxima fila de datos.



Estructura de una Aplicación JDBC

JDBC provee una interface que permite a los desarrolladores determinar qué tipo de datos es retornado, y obtener información sobre el dato mismo (meta data)

Para un objeto result set, una aplicación puede llamar al método getMetaData para retribuir datos meta descriptivos (por ejemplo, el número de columnas, sus nombres, y tipos de datos). [14] [15] [18]

5.9.- Conexión con una Base de Datos

Un objeto *Connection* representa una conexión con una Base de Datos. Una sesión de conexión incluye las sentencias SQL que son ejecutadas y los resultados que son retornados sobre la conexión. Una simple aplicación puede tener una o más

conexiones con una Base de Datos simple, o este puede tener conexiones con varias Base de Datos diferentes.

La forma estándar para establecer una conexión con una Base de Datos es llamar al método *DriverManager.getConnection* . Este método toma un string conteniendo una URL. La clase DriverManager, referida como la capa de manejo de JDBC, intenta localizar un driver que pueda conectarse a la Base de Datos representada por el URL. La clase DriverManager mantiene una lista de clases Driver registradas, y cuando el método getConnection es llamado, chequea cada driver en la lista hasta encontrar una que pueda conectarse a la Base de Datos especificada en la URL. El método de *Driver connect* usa este URL para realmente establecer la conexión

Un usuario puede saltear la capa de manejo de JDBC y llamar directamente a los métodos de Driver. Podría ser útil, en los raros casos que dos drivers pueden conectarse a la Base de Datos y el usuario quiera explícitamente seleccionar un driver particular.

El siguiente código ejemplifica abriendo una conexión a la Base de Datos ubicada en el URL "jdbc:odbc:wombat" con un identificador de usuario "oboy" y como password "12Java"

```
String url = "jdbc:odbc:wombat";
Connection con = DriverManager.getConnection(url,"oboy","12Java");
```

Una URL (Uniform Resource Locator) da información para localizar un recurso en Internet. Puede ser considerada como una dirección

La primer parte de URL especifica el protocolo usado para acceder a la información, seguido siempre por el signo dos puntos (:) . El resto de URL nos provee localización de la fuente de datos

El JDBC URL provee la manera de identificar una Base de Datos para que el driver apropiado la reconozca y establezca una conexión con ésta. Los usuarios no necesitan preocuparse acerca de cómo formar una JDBC URL. Simplemente usan un URL provisto de los drivers que está usando.

Así, los JDBC URL son usados con varias clases de drivers.

Primero, permiten diferentes drivers para usar diferentes esquemas para nombrar Base de Datos .

Segundo, permiten a los escritores de drivers codificar toda la información de conexión dentro de ellos.

Tercero, permiten un nivel de indirección. Esto significa que el JDBC URL pueda referirse a un host lógico o a un nombre de base de datos que sea dinámicamente traducida a un nombre real por el sistema de nombramiento de red.

La sintaxis estándar para los JDBC URL :

```
jdbc:<subprotocol>:<subname>
```

Tiene tres partes separadas por (:) :

- *el protocolo jdbc*: el protocolo en JDBC URL es siempre jdbc
- *<subprotocol>* : el nombre del driver o el nombre del mecanismo de conectividad a la Base de Datos, el cual puede ser soportado por uno o más

drivers. Un ejemplo es "odbc", el cual ha sido reservado para las URL que especifican los nombres de fuentes de datos ODBC-style.

- *<subname>* :manera de identificar la Base de Datos. Puede variar, dependiendo del subprotocolo, y puede tener un sub-subname con cualquier sintaxis interna del driver que elija. Una Base de Datos sobre un servidor remoto requiere mas información. Si al Base de Datos va a ser accedida sobre Internet , por ejemplo, las direcciones de red deberían ser incluidas en al JDBC URL como parte del subname y seguiría la convención de nombres URL estándar

```
//hostname:port/subname
```

Un manejador de driver puede reservar el nombre a ser usado como el subprotocolo en un JDBC URL. Cuando la clase *DriverManager* presenta este nombre a la lista de drivers registrados, el driver por el cual el nombre es reservado debería reconocerlo y establecer una conexión a la base que lo identifica. [14]

5.10.- Enviando Sentencias SQL

Una vez que la conexión es establecida, ésta es usada para pasar sentencias SQL. JDBC no pone ninguna restricción sobre la clase de sentencias SQL que pueden ser enviadas; brinda mucha flexibilidad, permitiendo el uso de las sentencias de la Base de Datos especifica o sobre sentencias no SQL. Esto requiere , sin embargo, que el usuario se haga responsable haciendo seguras las sentencias SQL enviadas.

Un objeto *Statement* es usado para enviar sentencias SQL a una Base de Datos. Actualmente JDBC provee tres clases de *objetos Statement* para enviar sentencias SQL a la Base de Datos, y tres métodos en la *interface Connection* crea instancias de esta clase. Estas clases y los métodos que lo crean son las siguientes:

- *Statement* , creados por el *método createStatement* , es usado para enviar sentencias simples SQL
- *PreparedStatement* , creado por el *método prepareStatement* , es usado por sentencias SQL que toman uno o más parámetros como argumentos de entrada (parámetros IN). PrepareStatement tiene un grupo de métodos que setean lso valores de los parámetros In, el cual son enviados a la Base de Datos cuando la sentencia es ejecutada. Las instancias de prepareStatement extiende de Statement y por lo tanto incluye métodos Statement.
- *CallableStatement* , creado por el *método prepareCall.* Son usados para ejecutar procedimientos almacenados SQL

La *interface Statement* provee métodos básicos para ejecutar sentencias y retribuir resultados.

Una vez que la conexión a una Base de Datos en particular es establecida, la conexión puede ser usada para enviar sentencias SQL. Un objeto Statement es creado con el método Connection.createStatement , como en el siguiente fragmento de código:

```
Connection con = DriverManager.getConnection(url,"sunny","");
Statement stmt = con.createStatement();
```

La sentencia SQL que sería enviada a la Base de Datos es suplantada como argumento de uno de los métodos para ejecutar un objeto Statement:

```
ResultSet rs = stmt.executeQuery("SELECT a,b,c FROM Table2");
```

La interface Statement provee tres diferentes métodos para ejecutar sentencias SQL, *executeQuery, executeUpdate*, y *execute.*

El *método executeQuery* es diseñado para sentencias que producen un simple conjunto de resultados, tal como sentencias SELECT.

El *método executeUpdate* es usado para ejecutar sentencias INSERT, UPDATE , DELETE y también sentencias SQL DDL como CREATE TABLE y DROP TABLE [14]

5.11.- DriverManager

La clase *DriverManager* es la capa de manejo de JDBC, que trabaja entre el usuario y los drivers. Este mantiene la pista de los drivers que están disponibles y establece una conexión entre una Base de Datos y su driver apropiado.

Para una simple aplicación, el único método que un programador general necesita usar directamente es *DriverManager.getConnection* . Como su nombre lo implica, este método establece una conexión a la Base de Datos.

La clase *DriverManager* mantiene una lista de clases Driver que se han registrado a sí mismas llamando al método *DriverManager.registerDriver.* Todas las clases Driver deberían ser escritas con una sección estática que crea una instancia de la clase y luego registra este con la clase *DriverManager* cuando este es cargado. Un usuario , normalmente, no debería llamar al *DriverManager.registerDriver* directamente; debería ser llamado automáticamente por el driver cuando es cargado. Una clase Driver es cargada, y por lo tanto, automáticamente registrada con el *DriverManager* llamado al método *Class.forName* . Este explícitamente carga la clase Driver. El siguiente código carga la clase acme.db.Driver:

```
Class.forName ("acme.db.Driver");
```

Si *acme.db.Driver* ha sido escrita para que su carga cause la creación de una instancia y también llama a DriverManager.registerDriver con esa instancia como parámetro (como debería hacerlo) entonces está en la lista de drivers de *DriverManager* y disponible para crear una conexión.

Una vez que las clases Driver han sido cargadas y registradas con la clase *DriverManager*, están disponibles para establecer una conexión con la Base de Datos. Cuando se hace el pedido de conexión con el método *DriverManager.getConnection* , el *DriverManager* testea cada driver para ver si puede establecer la conexión.

Puede suceder algunas veces que más de un driver JDBC es capaz de conectarse a un URL dado. En estos casos, *DriverManager* usaría el primer driver que encuentre que pueda conectarse satisfactoriamente al URL dado.

El siguiente ejemplo muestra todo lo necesario para setear una conexión con un driver tal como el driver JDBC-ODBC bridge. [14]

```
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
String url = "jdbc:odbc:fred";
```

```
DriverManager.getConnection(url,"userID","passwd");
```

5.12.- ResultSet

ResultSet contiene todas las líneas las cuales han sido satisfechas por las condiciones de las sentencias SQL, y provee acceso a los datos en aquellas líneas a través de un conjunto de métodos get que permiten acceder a varias columnas de la fila corriente. El método *ResultSet.next* es usado para moverse a la próxima fila del ResultSet, haciendo la próxima fila como fila corriente. [4] [12] [20]

5.13.- Base de Datos Oracle 7

Oracle ve a Java como el lenguaje de programación de opción para desarrollar aplicaciones distribuídas en el campo de Internet/Intranet. La Arquitectura de Computación de red Oracle es un marco para construír y desplegar dichas aplicaciones distribuidas.

Sin embargo, Oracle también reconoce que Java todavía no está listo para ser adoptado por grandes corporaciones debido al acceso a la Base de Datos relacional ineficiente . Para este fin, Oracle intenta llevar a Java a las empresas con una estrategia basada en tres iniciativas:

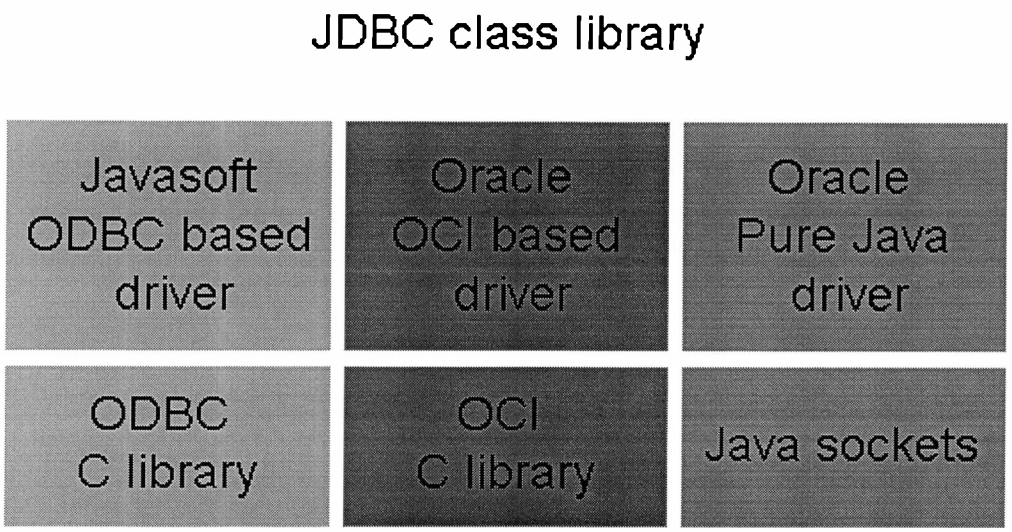
- Implementar diversos drivers JDBC optimizados para Base de Datos Oracle para brindar soporte a multiples configuraciones de aplicación;
- Introducir SQLJ como un estándar abierto para embeber al SQL en lenguaje Java, y
- Brindar soporte completo para la construcción de aplicaciones extensibles en Java a través de todas las arquitecturas de computación de red.

5.13.1.- Accesos a los datos relacionales desde Java

Oracle provee dos maneras simples por medio de las cuales los programadores de Java (escribiendo el código en cualquier arquitectura) puede acceder datos relacionales conveniente y eficientemente : JDBC y SQLJ. JDBC es una simple librería. Es relativamente de bajo nivel y puede ser tedioso para simples accesos SQL . SQLJ propone aumentar las capacidadades de JDBC brindandole a Java un acceso en tiempo de compilacion al esquema RDBMS , simplificando en gran medida el codigo brindando seguridad tipo y mayor rendimiento. SQLJ esta especificado como una integracion directa y completa de SQL y JAVA. Es un estándar propuesto por Oracle, IBM, Tandem, JavaSoft y otros.

5.13.2.- Arquitectura de JDBC de Oracle

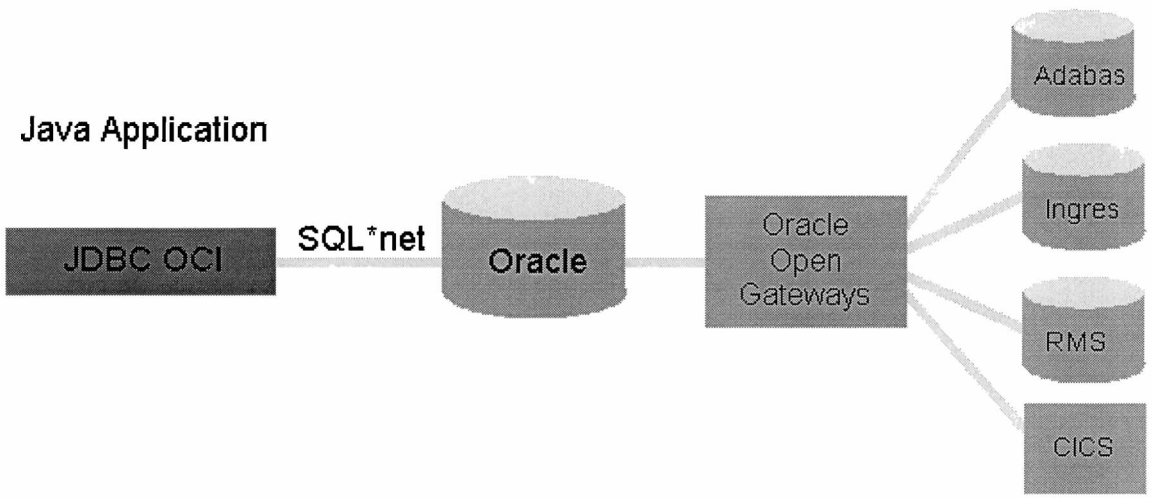
Oracle esta construyendo al menos dos drivers JDBC que brindan al usuario la flexibilidad para desplegar que sus aplicaciones Java en diferentes configuraciones. La arquitectura básica se muestra a continuación. La clase JDBC es compartida por el driver basado en OCI y el driver puro basado en Java. Además, ambos drivers comparten las mismas extensiones específicas Oracle 7 y Oracle 8 a la clase estándar JDBC



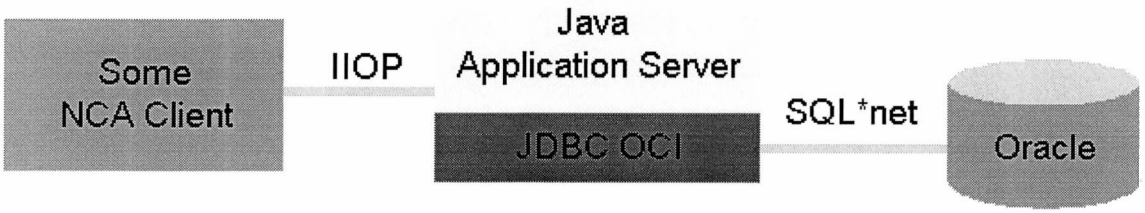
Diferentes Configuraciones Soportadas

Las aplicaciones de Java tradicionales Cliente/Servidor son soportados como se muestra a continuación

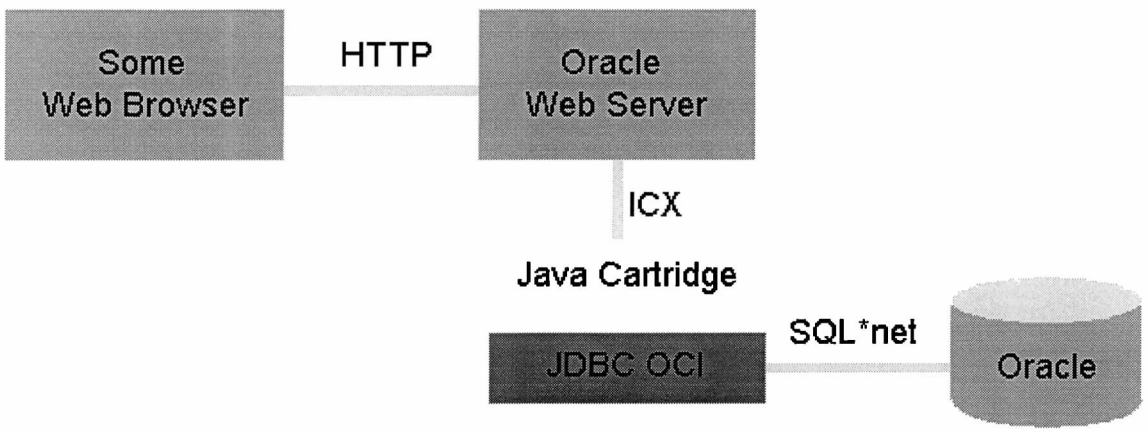
Esta configuración puede extenderse para acceder datos no tradicionales via Oracle Open Gateways como se muestra a continuación



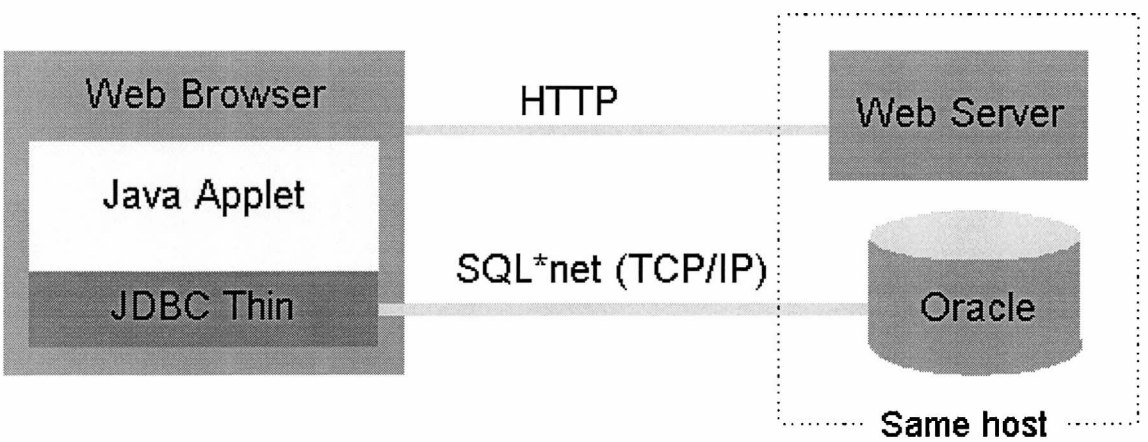
Los servers de aplicación capacitados para Java como el server de aplicación de la red Oracle pueden acceder a una base de datos Oracle via JDBC OCI como se muestra a continuación:



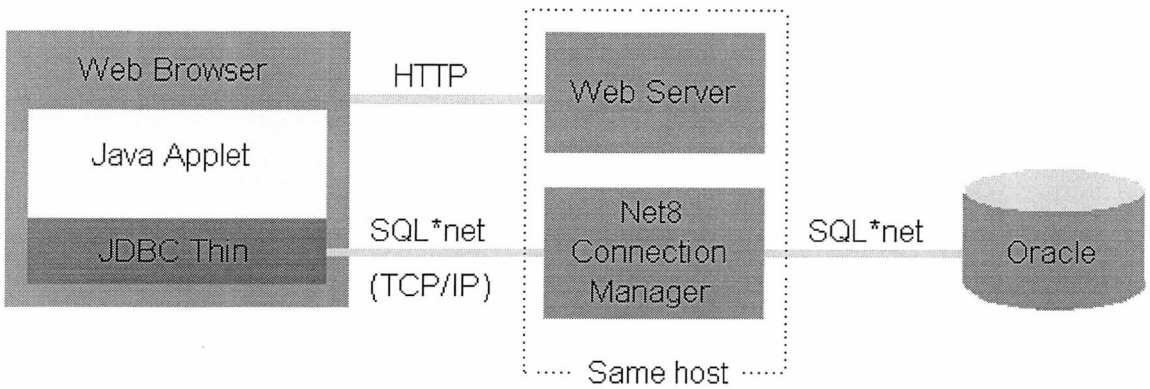
Permite al código que corre en un Cartbridge Java de Oracle acceder a una Base de Datos Oracle de la siguiente manera:



La configuración siguiente utiliza el 100% de la implementación de JDBC Java de Oracle para permitir a los applets de descarga acceder a la Base de Datos Oracle.



Alternativamente, *Connection Manager* de Oracle redirecciona los paquetes SQL*Net a la base de datos sobre un diferente host como se muestra abajo:



5.13.3.- Beneficios de JDBC de Oracle

- Una implementación completa del estándar : La implementación Oracle provee todos los puntos de entrada JDBC, incluyendo algunos de los detalles rechazados con frecuencia del streaming de columnas largas, conversiones de datos NLS, y soporte íntegro para la sintaxis de escape de ODBC SQL.
- Una variedad de plataformas: el driver JDBC de Oracle fue desarrollado con el JavaSoft de JDK, y portado hacia otro VMs de JAVA incluyendo el J++ de Microsoft, Symantec Café, y el VM de Oracle de acuerdo a su disponibilidad.
- Todos los tipos Oracle: Los APIs JDBCS estándar son naturalmente extendidos con nuevos puntos de entrada de conversion de datos que soportan los tipos Oracle7 (ROWID, MLSLABEL, variables de ligadura de CURSOR), todos los tipos Oracle8 (FILE,LOB,Collections), y todos los tipos de usuarios definidos (ADTs)
- Mejora de Producción : Oracle provee extensiones a JDBC para soportar operaciones en cantidad. Esto mejora mucho la producción de datos relacionados, especialmente para inserciones en cantidad.
- Dentro del RDBMS: Una version especialmente adaptada del driver JDBC que usa los puntos de entrada RDBMS de bajo nivel correrá directamente dentro del RDBMS. Proveerá el acceso directo más rapido a los datos SQL desde los procedimientos almacenados Java. [17] [15]

5.14.- SQLJ

SQLJ es una integración de sentencias SQL en programas Java.Es mas conciso que JDBC, y más ameno para el análisis estático y el chequeo de tipos. SQLJ utiliza la experiencia de Oracle con sus precompiladores para embeber SQL en muchos lenguajes del host (C, C++, Ada, Fortran, Cobol, Pascal), y en los programas PL/SQL almacenados en RDBMS. El preprocesador SQLJ es en sí un programa Java. Toma como entrada:

- Ya sea un archivo de cláusulas SQLJ las cuales son sentencias SQL anotadas opcionalmente con parámetros listas que declaran el nombre de los mapeos y el tipo de mapeos entre Java y SQL, o
- Un archivo de código de fuente Java en el cual las cláusulas SQLJ son embebidas.

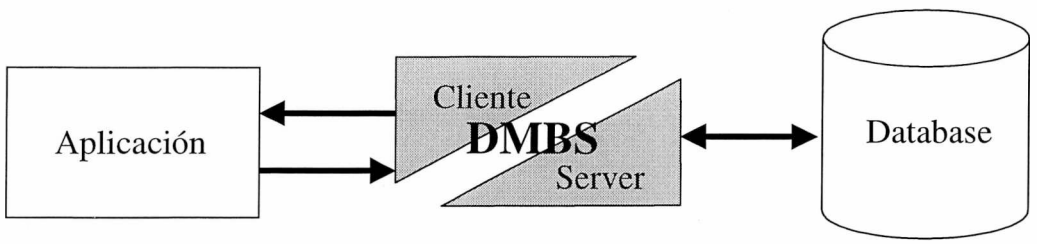
En ambos casos, el precompilar traduce las cláusulas SQLJ en clases Java que implementan las sentencias SQL especificadas. El sistema de tipo Java asegura entonces que los objetos de aquellas clases sean llamados con los números y tipos correctos de argumentos para pasar valores hacia y desde sentencias SQL y programas Java.[19]

Capítulo 6

RMI (Remote Method Invocation)

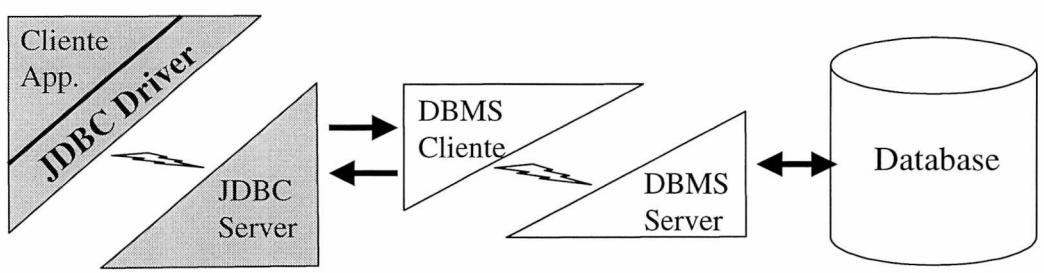
6.1.- Qué es RMI ?

Remote Method Invocation es el carácter distintivo en Java que permite crear aplicaciones distribuidas. Usando JDBC se pueden crear aplicaciones que son capaces de acceder a una Base de Datos tanto local como remoto. Es decir, se crean aplicaciones *stand-alone* o *two-tier.*



Arquitectura Two-Tier

Si queremos desarrollar aplicaciones sobre Inter o Intranet, lo que buscamos no son exactamente clientes importantes. Todos los pedidos de la Base de Datos están ruteados desde el cliente (applet) a un server JDBC que está conectado al cliente DBMS. En tal diseño todo SQL es ejecutado en el DBMS cliente y servidor. La aplicación cliente tiene que levantar sentencias SQL y procesar los resultados de la ejecución SQL. Todo dato SQL es comunicado al cliente DBMS.



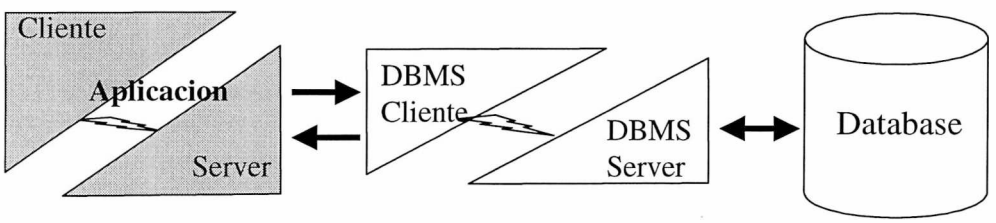
Arquitectura Three-Tier con Network JDBC Driver

Para minimizar el tamaño del applet, reducir la comunicación de red, y correr todos los procesos SQL al *two-tier*, necesitamos diferentes aproximaciones. Queremos

enviar pequeños applets sobre la red en vez de grandes aplicaciones, etiquetadas como applets.

Tenemos que dividir la aplicación en la parte del cliente y la del server. Del lado del servidor se procesan todos los accesos a los datos y envía solo objetos resultados consolidados al cliente. Del lado del cliente se maneja la presentación (GUI), validación de datos de entrada, y pedidos de comunicación a la aplicación server.

Tal diseño hace posible diseminar la aplicación sobre múltiples plataformas, tal como desktop PC's, servidor de aplicaciones y sistemas host para empresas.



Arquitectura Three-Tier con con Aplicación Cliente/Servidor

Tenemos que pensar cómo el cliente se comunica con el server. Una implementación es RMI, protocolo cliente-servidor basado en JAVA.

Los sistemas distribuidos requieren que los cómputos que corran en distintos espacios de direcciones, potencialmente en diferentes hosts, sean capaces de comunicar. El lenguaje Java soporta *sockets*, los cuales son flexibles y suficientes para una comunicación general.

Sin embargo los *sockets* requieren que el cliente y el server se relacionen en protocolos de niveles de aplicaciones para codificar y decodificar mensajes para intercambio, y el diseño de tales protocolos es problemático y tiende a error.

Una alternativa para sockets es *Remote Procedure Call* (RPC), el cual abstrae la interface de comunicación a nivel de una llamada al procedimiento. En vez de trabajar directamente con sockets, el programador tiene la ilusión de llamar a un procedimineto local, cuando de hecho los argumentos de la llamada son empaquetados y pasados al destino remoto de la llamada.

Los sistemas RPC codifican argumentos y retornan valores usando una representación de datos externa.

RPC, sin embargo, no se traduce bien en los sistemas de objetos distribuidos, donde la comunicación, entre objetos a nivel de programas que residen en distintos espacios de direcciones es necesaria.

Con el fin de cumplir con una semántica de una invocación a un objeto, los sistemas de objetos distribuidos requieren RMI (Remote Method Invocation). En tales sistemas, un objeto local *stub* administra la invocación sobre un objeto remoto.

6.2.- Para qué sirve RMI ?

RMI sirve para crear aplicaciones cliente/servidor distribuidos orientados a objetos real. Queremos que diferentes partes de nuestra aplicación corran sobre

diferentes computadoras y plataformas para comunicarse con la otra. Acá es donde RMI comienza a jugar.

Básicamente tenemos tres posibilidades para implementar procesamiento distribuido:

- Crear tu propia interface de comunicación con algún protocolo subyacente (TCP/IP, APPC)
- Object Request Broker (ORB)
- RMI para Java

Podemos considerar la primer posibilidad si pensamos en aplicaciones más pequeñas , con comunicaciones limitadas y definidas claramente que se necesitan entre las distintas partes de la aplicación

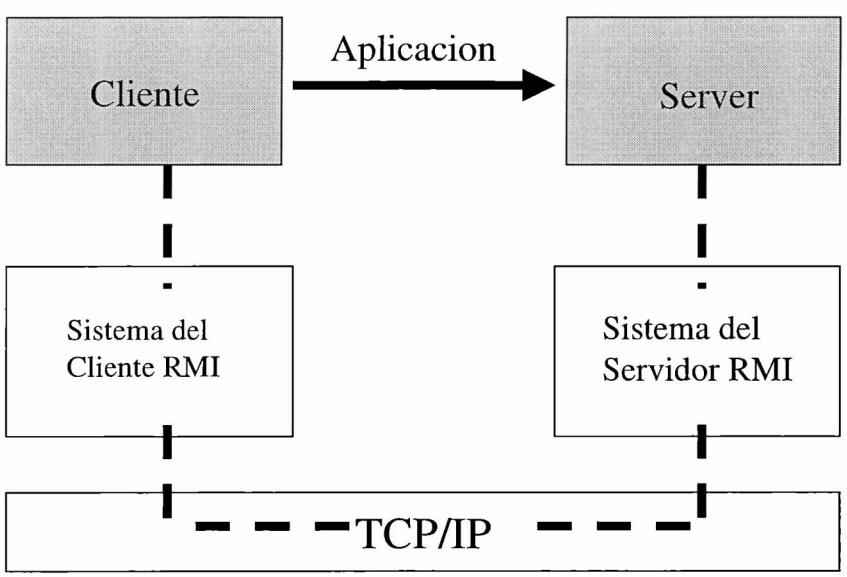
También, si evitar cualquier tipo de *overhead* es un problema, podemos considerar esta solución. Sin embargo, necesitaríamos poner mucha fuerza en el área de no-aplication para construir las interfaces, y cualquier cambio (aplicaciones, plataformas o comunicaciones) implicarían mayor fuerza para mantener la aplicación corriendo.

La segunda posibilidad, el ORB, es la solución si usamos diferentes lenguajes de programación, sobre diferentes plataformas, o si construimos una gran solución empresaria.

RMI es la solución para implementar comunicación entre objetos remotos en un ambiente Java puro.

Java provee todas las clases necesarias e interfaces para enviar mensajes desde un objeto a otro, aún si viven en diferentes computadoras.

6.3.- Cómo trabaja RMI ?



Desde el punto de vista conceptual, RMI habilita a la aplicación Java a llamar a un objeto Java remoto como si este estuviera en la máquina local.Mientras, desde el

punto de vista de la aplicación, un objeto invoca un método en otro objeto, RMI provee los servicios necesarios para localizar el objeto remoto, transfiere la llamada incluyendo todos los parámetros provistos a través de TCP/IP al server, invoca el método sobre el objeto server, y retorna el objeto a través del mismo path.

Uno de los servicios que ejecuta RMI, es enviar objetos sobre la red. Como TCP/IP no conoce de los objetos y solo conoce bits y bytes, los objetos tienen que ser convertidos a cadena de bytes. Esto no es muy dificultoso siempre y cuando estos objetos sean tipos de datos simples, tal como numéricos o strings. De hecho, esto fue ejecutado antes en una característica conocida como Remote Procedure Call (RPC).

RPC llama a los programas sobre computadoras remotas y transforma parámetros y retorna valores en bytes y viceversa. En un ambiente orientado a objetos, sin embargo, las cosas se vuelven un poco más complicado. Queremos enviar objetos, los cuales tienen propiedades que referencian aún más objetos, terminando en un gráfico de objetos referenciados. Convirtiendo tal gráfico de objetos en un stream de bytes y la recupera en una locación destino, esto es llamado *serialización* y *deserialización* . Si un objeto (o un gráfico de objetos) es escrito o leído desde un buffer, este proceso inclu ye serialización y deserialización y a menudo es llamado *marshaling* y *demarshaling* .

Parte de los servicios RMI es el marshaling y demarshaling de objetos. Sin embargo, cada objeto que está diseñado para usar estos servicios tiene la responsabilidad para asegurar que su información de estado puede ser transformado en un *byte stream.* Para asegurar esto, la definición de la clase objeto o de una de sus superclases tiene que incluir la interface *java.io.Serializable* . Esta interface no tiene métodos definidos, esto indica que los objetos de su clase pueden ser manejados por el readObject y writeObject métodos de clases stream I/O.

Si una clase implementa Serializable, todos los valores de la variable instancia de esa clase tienen que ser serializables. Esto significa que si una instancia apunta a otro objeto, la clase de este objeto tiene que ser Serializable también.

Los tipos de datos básicos Java son serializables tanto como todas las clases que representan tipos de datos, arreglos y vectores.

6.4.- Objetivos

Los objetivos para soportar los objetos distribuidos en lenguaje Java son:

- Soportar invocaciones remotas aparentes sobre objetos en distintas máquinas virtuales
- Soportar llamados de retorno desde los servidores a los applets
- Integrar el modelo de objeto distribuido al lenguaje Java de manera natural mientras retenga la mayoría de la semántica de los objetos del lenguaje Java
- Hacer que las diferencias entre el modelo de objeto distribuido y el modelo de objeto de Java local sean aparentes
- Hacer que las aplicaciones distribuidas confiablemente escritas sean tan simple como sea posible
- Preservar la seguridad provista por el ambiente de ejecución Java

Destacar todos estos objetivos es un requisito general que el modelo RMI sea tanto simple (fácil de usar) como natural (que se adapte bien al lenguaje)

Además, el sistema RMI debería permitir extensiones tales como garbage collection de objetos remotos, replicacion de servers y la activación de objetos persistentes para reparar a una invocación.

Estas extensiones deberían ser transparentes al cliente y agregar requerimientos mínimos de implementación por parte de los servidores que las usan. Para soportar estas extensiones, el sistema debería también soportar :

- Varios mecanismos de invocación : por ejemplo una simple invocación a un objeto simple o a un objeto replicado en múltiples locaciones. El sistema también deberá ser extensible a otros paradigmas de invocación.
- Varias semánticas de referencia para objetos remotos, por ejemplo referencias vivas (no persistentes), referencias persistentes y activación tardía.
- El ambiente seguro de Java provisto por los manejadores (administradores) de seguridad y cargadores de clases (class loaders)
- Garbage collection distribuido de objetos activos
- Capacidad de soportar transportes múltiples

En un modelo de objeto distribuido Java, un objeto remoto es aquel cuyos métodos pueden ser invocados desde otra máquina virtual Java, potencialmente sobre un host diferente. Un objeto de este tipo es descripto por uno o más interfaces remotas, las cuales son interfaces Java, que declara los métodos de un objeto remoto.

RMI es la acción de invocación a un método de una interface remota sobre un objeto remoto. Lo que es importante, una invocación de método sobre un objeto remoto tiene la misma sintaxis que una invocación de método sobre un objeto local.

6.5.- Modelo Distribuido y No Distribuido

Un modelo de objeto distribuido Java es similar a un modelo de objeto de la siguiente forma:

- Una referencia a un objeto remoto puede ser pasada como un argumento retornado como un argumento en cualquier invocación de método (local o remoto)
- Un objeto remoto puede ser adaptado (cast) a cualquiera del conjunto de interfaces remotas soportadas por la implementación que usa la sintaxis de Java interna para la adaptación (*casting*)
- El operador interno de Java *instanceof* puede ser usado para testear las interfaces remotas soportadas por un objeto remoto

El modelo de objeto distribuido de Java difiere del modelo de objeto de Java de las siguientes maneras:

- Los clientes de objetos remotos interactúan con interfaces remotas, nunca con las clases de implementación de aquellas interfaces
- Los argumentos no remotos para, y los resultados de una invocación de un método remoto son pasados por copia más que por referencia . Esto es debido a que las referencias a objetos son útiles de una máquina virtual simple

- Un objeto remoto es pasado por referencia, no copiando la implementación remota real
- La semántica de algunos de los métodos definidos por la clase *Object* son especializados para objetos remotos
- Debido a que los modos de falla de la invocación de objetos remotos son inherentemente más complicados que los modos de falla de invocación de objetos locales , los clientes deben tratar con excepciones adicionales que pueden ocurrir durante una invocación de método remoto

6.6.- Arquitectura del Sistema

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 trans porte 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).

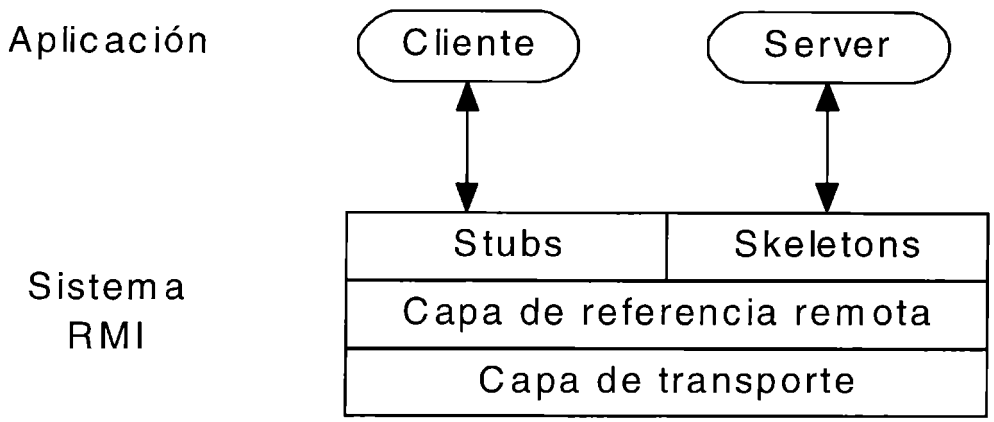
Otra técnica, llamada carga dinámica de *stubs*, es usada para soportar los stubs del lado del cliente, los cuales implementan el mismo conjunto de interfaces remotas que el objeto remoto mismo. Esta técnica, usada cuando un stub del tipo exacto ya no está disponible en el cliente, permite a éste usar los operadores predefinidos de Java para *casting* y chequeos de tipos.

6.6.1.- 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 conexiones 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 conexiones, 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.

6.6.2. 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 específicas 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.

6.6.3. 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 reconexión (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 conexión orientada a streams. La capa de transporte tiene cuidado de los detalles de implementación de las conexiones. Aunque las conexiones presentan una interface basada en *streams* se puede implementar un transporte de baja conexión por debajo de la abstracción.

6.6.4. Capa de transporte

En general, la capa de transporte del sistema RMI es la responsable de:

- Setear conexiones con los espacios de direcciones remotos.
- Manejar conexiones.
- Monitorear la vida de las conexiones.
- Escuchar las llamadas entrantes.
- Mantener la tabla de objetos remotos, que reside en el espacio de direcciones.
- Setear una conexión para una llamada entrante.
- Localizar al enviador de la llamada remota y pasarle la conexión.

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 conexión 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 conexiones entre el espacio de direcciones local y el remoto.
- Una conexión es la abstracción de las transferencias de datos (realizar entrada/salida)
- La abstracción transporte maneja canales. Cada canal es una conexión 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 conexiones entrantes del espacio de direcciones, setear un objeto conexión 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.

6.6.5.- Seguridad

En Java, cuando un cargador de clases las carga desde el CLASSPATH local, se considera que son clases confiables y no están 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 carguen desde fuentes confiables (por ejemplo el host del applet) y no intenten acceder a funciones sensibles.

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.

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.

6.6.6.- 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.

6.6.6.1. 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 protegerán. 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.

6.6.6.2. 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 conexiones 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.

6.6.6.3. 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.

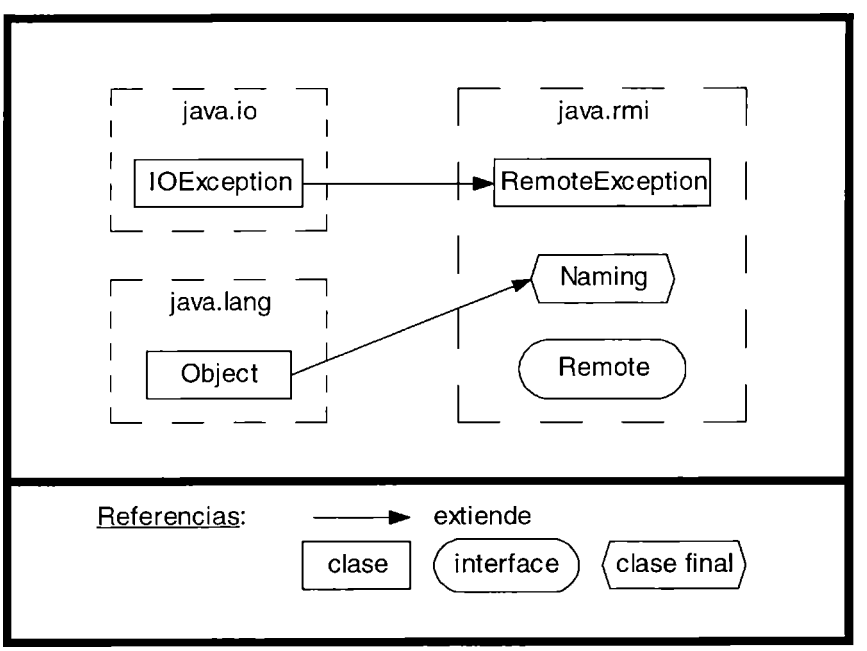
Para habilitar la carga desde una fuente de red, el host donde residen las clases debe haberse configurado con la propiedad java.rmi.server.codebase. Esto permite al sistema RMI embeber el URL de la clase en la forma serializada de la clase.

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 protegerse contra esto 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.

6.6.7. 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.



6.6.7.1. 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.

Todas las interfaces remotas extienden, directa o indirectamente, la interface java.rmi.Remote. La interface Remote no define métodos:

```
public interface Remote { }
```

Los métodos en una interface remota deben ser definidos como sigue:

- Cada método debe declarar *java.rmi.RemoteException* en su cláusula *throws*, además de cualquier excepción de aplicación específica
- Un objeto remoto pasado como un argumento o valor de retorno debe ser declarado como una interface remota, no la clase de la implementación.

6.6.7.2. 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. Para asegurar la robustez de aplicaciones que usan el sistema RMI, cada método declarado en una interface remota debe especificar *java.rmi.RemoteException* en su cláusula throws.

6.6.7.3. 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.

6.6.8. Interfaces del Server

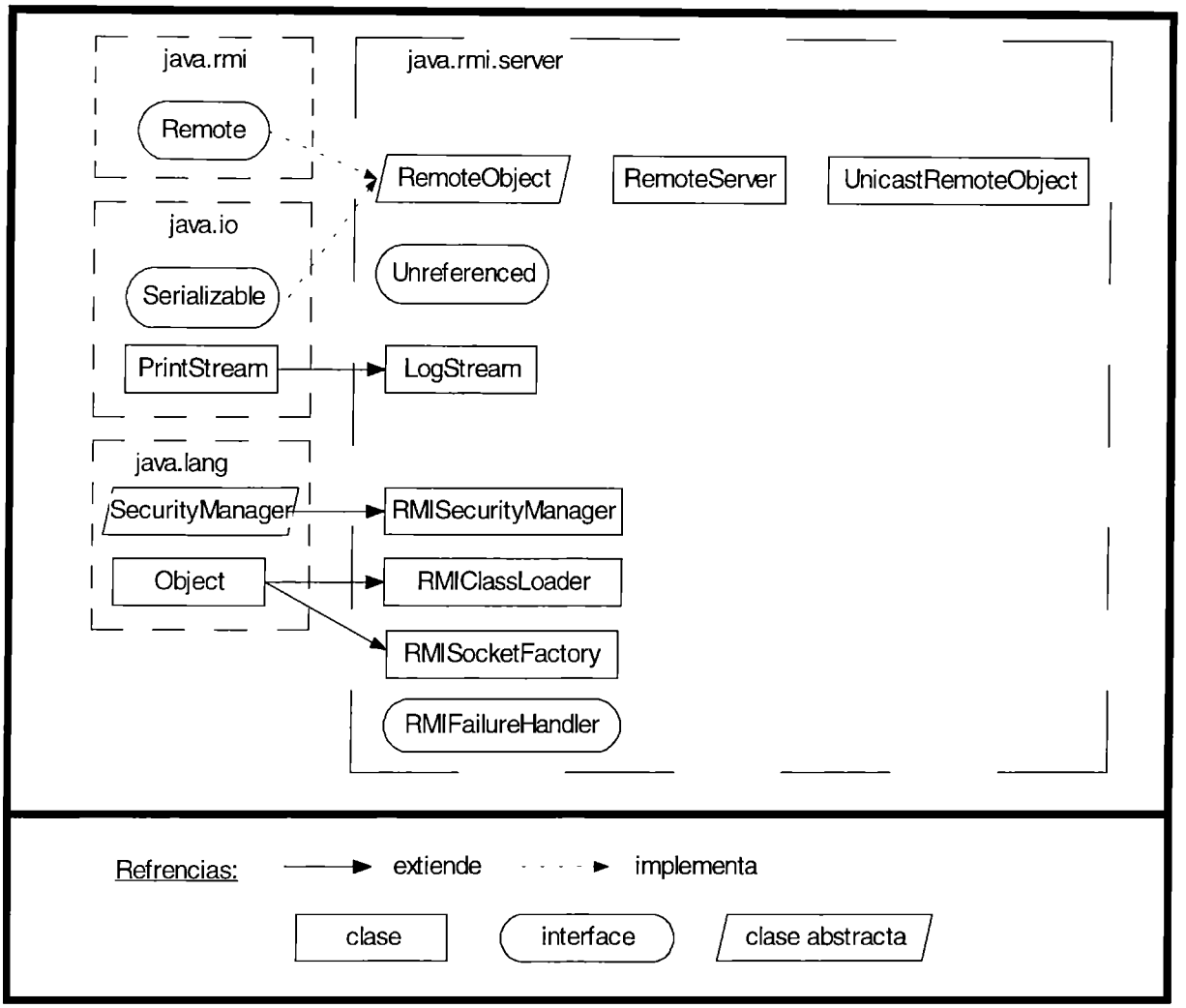
Cuando se implementa un server, las interfaces del cliente están disponibles y son extendidas con aquellas que permiten la definición, creación y exportación de objetos remotos.

6.6.8.1. Clase RemoteObject

La clase *java.rmi.server.RemoteObject* implementa el comportamiento de java.lang.Object para los objetos remotos.

6.6.8.2. Clase RemoteServer

La clase *java.rmi.server.RemoteServer* es la superclase común de todas las implementaciones server y provee el ambiente para soportar un amplio rango de semánticas de referencias remotas. En este momento, la única subclase que soporta es *UnicastRemoteObject.*



6.6.8.3. Clase UnicastRemoteObject

La clase *java.rmi.server.UnicastRemoteObject* provee soporte para referencias a objetos activos punto a punto usando streams basados en TCP. La clase implementa un objeto server remoto con las siguientes características:

- Las referencias son válidas a lo sumo durante la vida del proceso que crea el objeto remoto.
- Se usa transporte basado en conexiones TCP.
- Las invocaciones, parámetros y resultados usan un protocolo de streams para comunicarse entre clientes y server.

6.6.8.4. 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 vacíe el conjunto. Los objetos remotos son recolectados únicamente cuando no existan más referencias, ya sean locales o aquellas mantenidas por los clientes.

6.6.9. 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.*

6.6.10. 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.

6.6.11. 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.

6.6.12. 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.

6.6.13. 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.

6.6.14. Compilador Stub y Skeleton

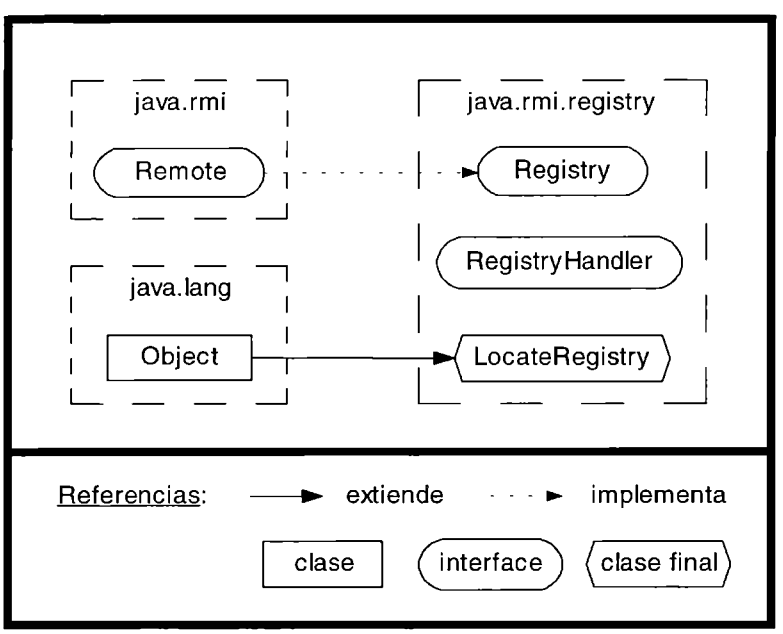
El compilador de stubs y skeletons *rmic* es usado para compilar los stubs y skeletons apropiados para una implementación de objeto remoto específico. El compilador es invocado con el nombre de la clase del objeto remoto calificado por el paquete al cual pertenece la clase. La clase debe haber sido compilada con éxito previamente.

6.7. 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.



6.7.1. 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.

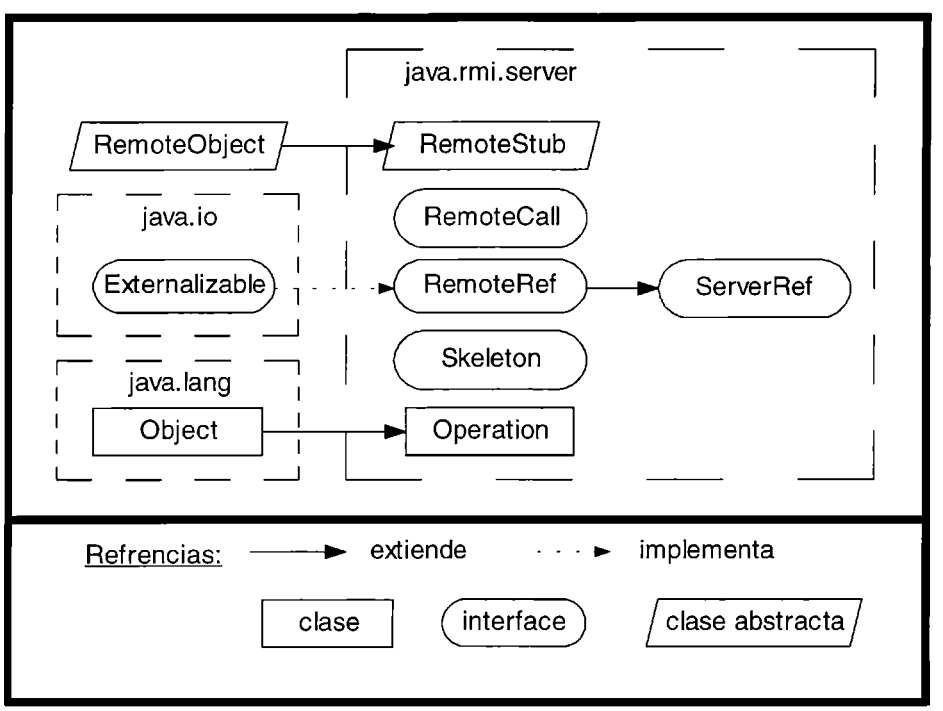
6.7.2. Clase LocateRegistry

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.

6.7.3. Interface RegistryHandler

RegistryHandler sirve como interface para la implementación privada.

6.8.- Interfaces Stub/Skeleton



6.8.1.- 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.

6.8.2.- 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.

6.8.3.- 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.

6.8.4.- Interface ServerRef

La interface *ServerRef* representa el manejador, del lado del server, de la implementación de objetos remotos.

6.8.5.- 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.

6.8.6.- Clase Operation

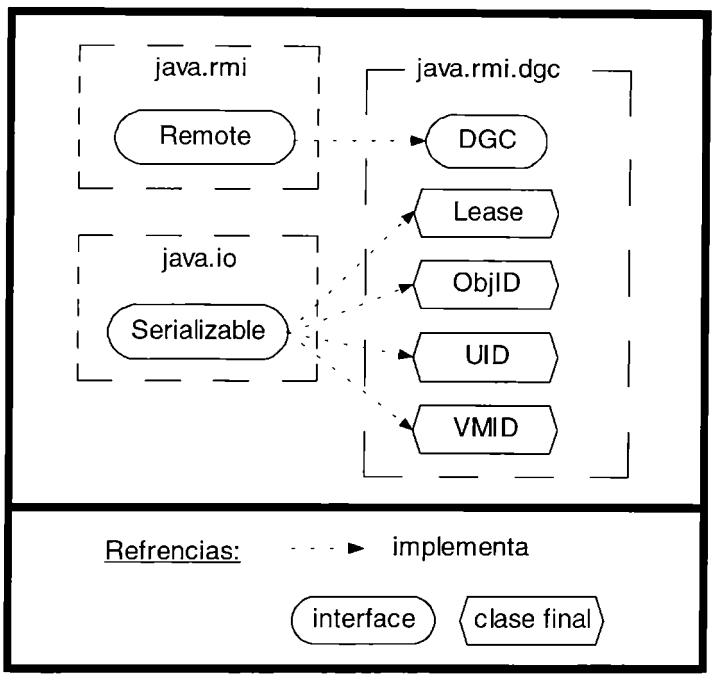
La clase *Operation* mantiene una descripción de un método Java para un objeto remoto.

6.9.- Interfaces del recolector de residuos

6.9.1.- 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.



6.9.2.- 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.

6.9.3.- 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.

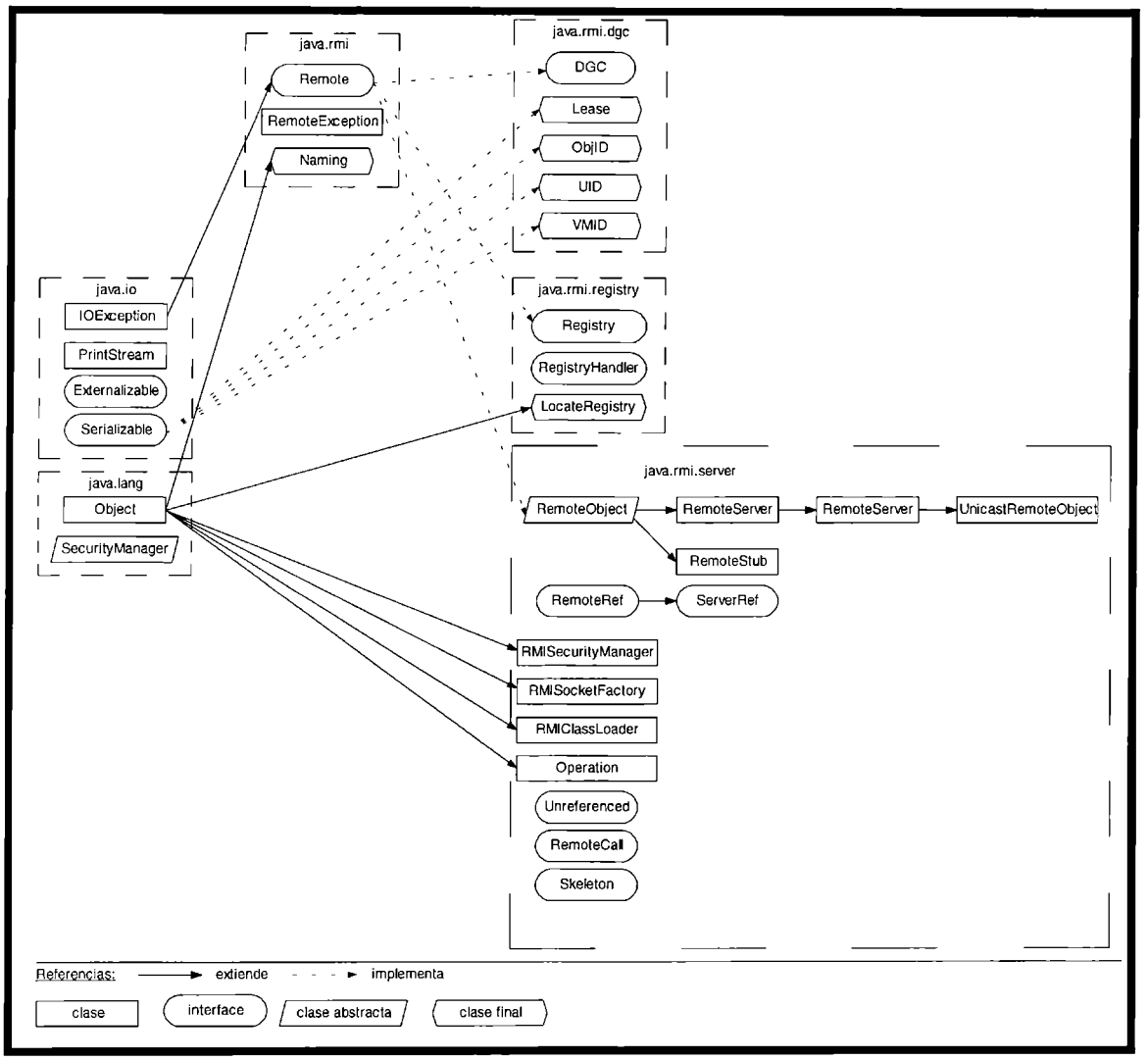
6.9.4.- Clase UID

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.

6.9.5.- 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. [4] [14] [16]

6.10.- Gráfico general de RMI



Capítulo 7

Desarrollo e Implementación del Protocolo Kerberos

7.1.- Objetivos

Realizar procesos para restringir el acceso a los servicios para los distintos perfiles de usuario. Dichos procesos estarán basados en el Modelo de Autenticación y Autorización Kerberos. Será implementado sobre una aplicación realizada en Lenguaje Java.

La aplicación integrará servicios con una interface homogénea y susceptible de la definición de perfiles de usuarios para los cursos a tomar para realizar una carrera del Departamento de Informática de la Facultad de Ciencias Exactas de la UNLP.

7.2.- Introducción

Al comenzar a investigar para esta tesis tuvimos que estudiar todo lo relacionado con seguridad en redes, ya que en la actualidad es un tema a resolverse porque Internet día a día cobra mayor dimensión, a demás nuestra tesis se basa en el análisis e implementación de un protocolo de seguridad para aplicaciones distribuidas. Este protocolo es el modelo de autenticación y autorización Kerberos explicado en el capítulo 3, que permite otorgar seguridad a aplicaciones distribuidas, que son utilizadas por usuarios fuertemente autenticados en una red. Kerberos se basa en la autenticación de usuarios y autorización de los mismos para que puedan usar los servicios requeridos mediante el intercambio de credenciales encriptadas y el uso de claves de sesión.

Podríamos hacer la siguiente similitud, para entender como trabaja Kerberos:

- Cómo Kerberos permite autenticarse a usted mismo?
- Cómo se autentica usted mismo en la vida real?

Típicamente, usted muestra su documento de identidad o su licencia de conducir. Usualmente la licencia no puede falsificarse y consiste de una foto y datos como nombre, dirección, etc. Dicha identificación tiene un tiempo de vida limitado, representada por la fecha de expiración, la cual debe ser renovada, para que siga siendo válida. Hay un organismo (el que emite la licencia o el carnet) que ha enlazado una identidad dada a una apariencia física. Esta apariencia física usualmente consiste en una foto o algunos datos físicos y se la considera inimitable (esto significa que uno no se puede cambiar a si mismo para parecerse a alguien diferente)

Kerberos trabaja básicamente de la misma manera. Es típicamente usado cuando un usuario sobre una red quiere usar un servicio de la red, y el servicio quiere asegurarse que el usuario es quien dice ser. Para esto, el usuario presenta un ticket que fue impreso por el server de autenticación Kerberos (AS), tal como la licencia de conducir emitida por la dirección que otorga los permisos de conducir. El servicio luego examina el ticket para verificar la identidad del usuario. Si los chequeos realizados son validos , luego el usuario es aceptado y puede usar el servicio.

Notar que la demostración de identidad esta limitada a un número de casos. Antes que nada la tarjeta no puede ser arruinada (como por ejemplo, cambia la fecha de nacimiento o la foto). En segundo lugar la persona que realiza la autenticación debe aceptar al organismo que emitió el carnet. Hay algunas otras preocupaciones como por ejemplo, que la persona que esta siendo autenticada realmente no ha cambiado en apariencia o nombre, o que el carnet sea robado, etc.

Al comenzar nuestra tesis se planteo la posibilidad de integrar la implementación del protocolo Kerberos con una aplicación kerberizada que permitiría a distintos usuarios acceder a distintos servicios del Departamento de Informática de la Facultad de Ciencias Exactas de la U.N.L.P..

Por el motivo mencionado anteriormente analizamos:

- Los distintos perfiles de usuarios que pueden acceder en forma remota a los distintos servicios brindados por la facultad.
- Los servicios con sus restricciones de uso para los diferentes tipos de usuarios.
- El diseño de la base de datos con sus tablas

Luego analizamos como la aplicación iba a interactuar con Kerberos y decidimos que cada vez que un usuario quiere acceder a un servicio debe pedir el servicio, entonces la aplicación llamará a un método remoto "seguridad de Kerberos" y este método otorgará la clave de acceso al servicio deseado si el usuario fue autenticado y autorizado satisfactoriamente.

Finalmente analizamos e implementamos el protocolo de seguridad Kerberos en el lenguaje Java.

Queremos destacar que el desarrollo de nuestra tesis es una implementación en lenguaje JAVA y no una simulación y que hicimos una pequeña aplicación kerberizada para mostrar su buen funcionamiento.

7.3.- Requerimientos

Cualquier usuario que desea utilizar el Protocolo de Autenticación y Autorización Kerberos implementado en nuestra tesis, para otorgar seguridad a la red, necesita disponer de:

- Una aplicación Kerberizada
- Los paquetes de JAVA que permiten utilizar RMI
- Java Virtual Machine para correr la aplicación.
- La Base de Datos Kerberos deberá constituirse a partir de la Base de usuarios registrados para la red

Conexión a un Server que contenga:

- Protocolo Kerberos corriendo y la base Kerberos con los usuarios autenticados.
- Drivers JDBC-ODBC correspondientes para acceder a la base Kerberos.

- Paquetes de JAVA para el manejo de RMI.

7.4.- Análisis y Diseño del Protocolo Kerberos

7.4.1.- Propuesta

Se propone una aplicación cuyo objetivo es integrar distintos servicios con una interface homogénea y susceptible de la definición de perfiles de usuarios implementada en lenguaje Java por distintos grupos de alumnos.

Se nos asignó la tarea de Brindar seguridad a la aplicación mediante el protocolo de autenticación y autorización Kerberos.

7.4.2.- Comienzos del desarrollo de la propuesta

Al comenzar las especificaciones correspondientes, surgió la inquietud de conocer como sería integrada la aplicación.

El director del proyecto determinó que realizáramos el análisis de la aplicación en forma conjunta con el grupo que implementará la interface para brindar el acceso de los alumnos a los distintos servicios, debido a la necesidad de integración de la aplicación propuesta.

Comenzamos realizando las primeras definiciones de la estructura de la Base de Datos y los Servicios autorizados que brindará la aplicación. Dado que la misma será realizada en el Lenguaje Java, el modelo Kerberos será codificado en dicho lenguaje.

Una vez aceptadas las definiciones propuestas continuamos con la investigación del lenguaje. Dado que el mismo no contaba con un ambiente de desarrollo, en la Facultad nos facilitaron el software Visual Age for Java, y comenzamos a interiorizarnos en el uso de esta herramienta, siguiendo las pautas establecidas en el menú de ayuda.

También estudiamos cómo realizar los accesos a las Bases de Datos, mediante el uso del API JDBC e investigamos el funcionamiento de RMI (Invocación de Métodos Remotos) para los accesos remotos a los procesos que implementamos de Kerberos que deberán ser accedidos por todos los usuarios que quieran usar un servicio, mediante una aplicación kerberizada.

7.4.3.- Especificaciones de la Aplicación

Como mencionamos anteriormente la aplicación permitirá a distintos perfiles de usuarios acceder en forma remota a distintos servicios brindados por la Facultad de Ciencias Exactas.

Servicios

Generales	Específicos según Distintos Clientes
Ftp	1. Consultar información de cursos
IRC	2. Consultar información de materias
Mail	3. Registración/Inscripción de Clientes
Monografías	4. Armar Calendario
News	5. Consultar Trabajos

	6. Depositar Trabajos
	7. Consultar Bibliografía
	8. Depositar Bibliografía
	9. Consultar Calificaciones
	10.Depositar Calificaciones
	11.Estadísticas

Quienes utilizarán los servicios y que trabajos realizarán

Tipos de Clientes	Tipos de Trabajos
Administrativos	Exámenes
Alumnos	Trabajos Prácticos
Graduados	Monografías
Personas en General	
Profesores	

Descripción de los servicios definidos por el usuario y alcance de los mismos

1. Consultar Información de Carreras
   Los servicios que se enumeran a continuación pueden ser accedidos por cualquier tipo de cliente.

- *Calendario*
- Se expondrá lugar, fecha y hora de cursada para cada materia que conforma la carrera.
- *Profesores que lo dictan*
- Permite obtener todos los profesores que conforman la carrera, incluyendo todas las materias.
- *Valor del curso*
- Importe total de la carrera.
- *Materias que lo componen*
- Detalle de las materias que forman la carrera.
- *Cupo*
- Vacantes por materia.
- *Requisitos*
- Requisitos que debe cumplir el alumno para poder realizar la carrera.

2. Consultar información de materias

- *Horario de Cursada*
  Se expondrán fechas de inicio, días y hora en que se cursa la materia, entre otros.
  Este servicio puede ser efectuado por cualquier tipo de cliente.
- *Profesor que la dicta*

Nombre del profesor que dicta la materia.
Este servicio puede ser requerido por cualquier tipo de cliente.

- *Bibliografía*
  Material de estudio para la materia.
  Sólo autorizado para profesores, alumnos y graduados del curso al cual pertenece la materia.
- *Información de Trabajos*
  Se expondrán los trabajos prácticos, las monografías a presentar.
  Sólo autorizado para profesores y alumnos que cursan la materia.
- *Calendario*
  Se expondrán fechas de inicio, fechas de entrega de trabajos.
  Sólo autorizado para profesores y alumnos del curso al cual pertenece la materia.

3. Registración / Inscripción de Clientes

- *Registración de Clientes*
  Este servicio puede ser efectuado por personas en general. Al efectuarse la registración el cliente se registra como uno de los tipos definidos anteriormente (alumno, profesor, ...).
  Para registrarse se solicitan los datos personales y una password de acceso, la cual será utilizada por el cliente para acceder a los servicios para los cuales estará autorizado, según su nivel de cliente.
- *Inscripción en las materias como profesor*
  Este servicio está autorizado sólo para personas que se encuentran registradas como clientes. (Si no es cliente, deberá realizar una registración previa).
- *Inscripción en los cursos como alumno*
  Este servicio está autorizado sólo para personas que se encuentran registradas como clientes. (Si no es cliente, deberá realizar una registración previa).
- *Inscripción en los cursos como administrativo*
  Servicio está autorizado para clientes. (Si no es cliente, deberá realizar una registración previa).

4. Armar Calendario

Este servicio está autorizado para administrativos y profesores de la materia. Se armará por curso, y por materia. Contendrá fechas, lugares, horarios, profesores que dictan los cursos, entre otros datos.

5. Consultar Trabajos

- *Consultar Monografías*
  Se encuentra al nivel de cualquier otro servicio predefinido (mail, news). Requerido por cualquier tipo de cliente.
- *Trabajos Prácticos Aprobados*
  Se expondrán los trabajos prácticos aprobados. Los deposita el profesor. Pueden acceder alumnos que cursan la materia y los profesores.

- *Trabajos Prácticos a realizar*
  Se expondrán los trabajos prácticos.
  El acceso está permitido para alumnos que cursan la materia y profesores.

6. Depositar Trabajos

- El alumno entrega trabajos prácticos, monografías, exámenes de su materia.
- Los profesores depositan trabajos, que pueden ser luego accedidos por distintos niveles.
- Si deposita un trabajo aprobado es accesible para los alumnos que cursan la materia, mientras que si el Trabajo está desaprobado sólo puede accederlo el autor. El profesor deposita además los TP a realizar.

7. Consultar Bibliografía

- Se encuentra accesible para profesores, alumnos y graduados de la carrera. Brinda información del material bibliográfico de las materias de dicha carrera.
- Consultar papers (para alumnos, profesores y graduados).

8. Depositar Bibliografía

- Depositar papers.

9. Consultar Calificaciones

- *Consultar Notas de Alumnos del Curso*
  Los profesores pueden consultar las notas de todos sus alumnos.
  Los alumnos sólo sus notas (parciales, finales).
- *Consultar Notas del Alumno*
  Los administrativos y los profesores pueden consultar las notas de los alumnos (Certif. Analítico).
  Los alumnos pueden consultar sus propias notas.

10. Depositar Calificaciones
    Los pueden depositar los administrativos y profesores.

11. Estadísticas
    Accesibles para administrativos y profesores.

- *Estadísticas de Alumnos*
  Se permitirán estadísticas del tipo:
  - Alumnos aprobados por curso,
  - Inscriptos en un curso,
  - Alumnos que entregaron los trabajos en término,
  - Alumnos que consultaron determinadas páginas.
- *Estadísticas de Cursos*
  Se permitirán estadísticas del tipo:
  - Alumnos inscriptos en un curso,

- Cantidad de aprobados y desaprobados en un curso,
- Cursos con mayor asistencia,
- Cursos dictados.

• *Estadísticas por Servicios*
Se permitirán estadísticas del tipo:
- Cuantos clientes utilizaron un Servicio determinado.
- Cuál es el Servicio más utilizado.
- Porcentajes de uso de un servicio, según tipo de clientes.

7.4.4.- Definición de la Base de Datos de la Aplicación

• Tabla de Descripción de Servicios:

Nombre Campo	Descripción
Id_servicio	Identificación del Servicio
Descripción	Descripción del Servicio

• Tabla Tipo-Clientes:

Nombre Campo	Descripción
Id_Cliente	Identificación del Cliente
Tipo de cliente	Tipo de cliente (alumno,prof., administrativo)

• Tabla de Datos de Clientes

Nombre Campo	Descripción
Id_cliente	Identificación del cliente
Datos personales	DNI, Nombre, dirección,teléfono, fax, e-mail
e_mail	Dirección de e_mail.
Tipo de Cliente	Tipo de Cliente (alumno, prof.)

• Tabla de Carreras

Nombre Campo	Descripción
Id_carrera	Identificación de la carrera
Id_plan	Identificación del Plan de estudios
Titulo	Nombre de la carrera
Valor	Importe Total del curso
Año del Plan	Año del Plan de estudios para dicha carrera
Duración	Duración del curso.
Incumbencias	Incumbencias
Vigente (S/N)	Especifica la vigencia del plan

• Tabla de Plan de estudios (conjunto de materias)

Nombre Campo	Descripción
Id_plan	Identificación del plan de estudios

Id_materia	Identificación de la materia
Nombre	Nombre de la materia
Correlativas	Id de las materias correlativas
Anual/Semestral	Si la materia es anual/semestral
Bibliografía	Bibliografía de la materia
Cupo	Cupo de alumnos por materia
Duración	Cantidad en horas

- Tabla Graduados

Nombre Campo	Descripción
Id_cliente	Identificación del Cliente
Id_carrera	Identificación de la carrera
Fecha Graduación	Fecha de Graduación
Comisión Evaluadora	Profesores que evaluaron al graduado

- Tabla Materia-Profesor

Nombre Campo	Descripción
Id_cliente	Identificacion del profesor
Id_materia	Identificación de la materia
Año dictada	Año en que se dicto la materia con dicho profesor
Cargo	Ayudante, Titular, Adjunto.

- Tabla de Materia-Alumno:

Nombre Campo	Descripción
Id_cliente	Identificacion del Alumno
Id_materia	Identificación de la materia
Año cursada	Año de cursada (por si la materia se dicta <> en <> año
Situación	Auxiliar,titular
Estado cursada	Si aprobo la cursada o no
Estado final	Si aprobo el final o no
Notas finales	Notas de los finales (aprobados y desaprobados de la materia)

- Tabla de Calendario de Conferencias

Nombre Campo	Descripción
Id_conferencia	Identificación de la conferencia.
Id_cliente	Identificación del profesor que dicta la conferencia.
Fecha inicio	Fecha de inicio de la conferencia.
Fecha fin	Fecha de finalización de la conferencia.
e_mail	Dirección de mail donde registrarse.

- Tabla de Trabajos a Realizar

Nombre Campo	Descripción
Id_Materia	Identificación de la materia.
Id_trabajo	Identificación del trabajo.
Tipo	Tipo de trabajo (Trabajo práctico, monografía, examen).
Tema	Tema que trata el trabajo.
Especificaciones	Especificaciones del trabajo
e_mail	Dirección de mail donde se encuentra el trabajo.

- Tabla de Trabajos Entregados

Nombre Campo	Descripción
Id. Materia	Identificación de la Materia
Id_trabajo	Identificación del trabajo.
Id_cliente	Identificación del cliente.
Calificaciones	Calificaciones del trabajo (Aprobado o no).

- Tabla de Calendario Trabajos (por carrera, materia para cada trabajo)

Nombre Campo	Descripción
Id_materia	Identificación de la materia.
Id_trabajo	Identificación del trabajo.
Fecha	Fecha límite para entrega de trabajo o fecha de examen.
Hora	Hora límite para entrega de trabajo o hora de examen.

- Tabla de Calendario Materias (por materia y por carrera para cada trabajo)

Nombre Campo	Descripción
Id_materia	Identificación de la materia.
Horario Teoria	Horario en el cual se dictará la teoría
Horario Práctica	Horario en el cual se dictará la práctica
Fecha de inicio	Fecha de inicio de la materia

- Tabla de Cuotas - Carrera

Nombre Campo	Descripción
Id_carrera	Identificación de la carrera
Id. Cuota	Número de cuota
Valor	Importe de la cuota

- Tabla de Cuotas Pagas

Nombre Campo	Descripción
Id_carrera	Identificación de la carrera
Id. Cuota	Número de cuota
Id_Cliente	Identificación del cliente

7.4.5.- Especificaciones de Kerberos

Kerberos es un Modelo de Autenticación y Autorización que otorga seguridad y confibilidad a la red de redes, permitiendo que usuarios autenticados y autorizados accedan a los servicios.

Para ello cada usuario envia a Kerberos un pedido para usar un servicio ingresando los siguientes datos:

- Id. Usuario
- Tipo de Usuario
- Materia
- Id. Servicio
- Direccion de la computadora desde donde se quiere usar el servicio
- Ticket TGS anterior
- Ticket de Servico anterior

Kerberos recibe estos datos, los procesa y devuelve la clave de acceso al servicio o un mensaje explicando porque no puede usar el servicio.

Este modelo cuenta con tres procesos principales que son:

- *AS*
  Determina que el usuario ingresado es auténtico, verificando la existencia del mismo, luego si el tiempo de vida del ticket TGS anterior expiró genera un nuevo ticket para acceder al TGS, en caso contrario accede con el anterior porque Kerberos permite la reusabilidad de los ticket mientras su tiempo de vida no haya concluido, también otorga una clave de sesión para los futuros intercambios entre el cliente y el TGS.
- *TGS*
  Recibe la identificación del servicio pedido, el ticket TGS encriptado con su clave y un autenticador encriptado con la clave de sesión. Al ticket lo desencripta con su clave y obtiene la clave de sesión para desencriptar al autenticador. Compara los datos del autenticador y el ticket, si estos coinciden quiere decir que el cliente es aquel para el cual se imprimió el ticket entonces verifica si el ticket anterior del servico es para el servicio pedido y si su tiempo de vida no expiró lo utiliza sino arma un nuevo ticket para acceder al servicio deseado. Otorga también una nueva clave de sesión que comparten el server de servicio y el cliente para realizar sus futuros intercambios.
- *Server de Servicios*
  Recibe el ticket para el servicio, generado por el TGS encriptado con su clave y el autenticador, generado por el cliente encriptado con la clave de sesion, los desencripta y compara sus datos si estos coinciden verifica que el usuario este autorizado a usar el servicio pedido de ser asi devuelve la clave de acceso al mismo para que pueda usar el servico y dos listas una que contiene el ticket TGS y otra que contiene el Ticket de Servicio, para que los pueda reusar en futuros pedidos.

En este modelo los tickets están encriptados con la clave del proceso que los recibe garantizando así que serán los únicos que los podrán desencriptar. El ticket TGS esta encriptado con la clave del TGS y el ticket de servicio con la clave del mismo.

En caso que un impostor tome un ticket que no ha sido impreso para él, como para usarlo también tiene que enviar un autenticador de usuario, los procesos al comparar los datos del ticket y el autenticador notarán que no matchean y no tomarán al usuario como un cliente auténtico.

Todo este mecanismo garantiza que los servicios serán usados por usuarios autenticados y autorizados.

Recién una vez que se verifica que el cliente es auténtico en el server de servicio se verificará si el cliente esta autorizado a usar el servicio pedido. Determinamos que la verificación de la autorización se haga en cada server de servicios, por tal motivo puede suceder que el TGS imprima un ticket para un cliente auténtico pero que no este autorizado a usar el servicio pedido. En este caso será el server de servicios quien no habilite al usuario a usar el servicio.

7.4.6.- Definición de la Base de Datos de Kerberos

- Tabla de Clientes:

Nombre Campo	Descripción
Id_cliente	Identificación del cliente
Clave	Clave que utilizaría Kerberos
Datos de expiración	Fecha, hora de finalización de la clave

- Tabla de Servicios:

Nombre Campo	Descripción
Id_servicio	Identificación del Servicio
Clave	Clave de Identificación
Clave de Acceso	Clave de acceso al servicio
Datos de expiración	Tiempo de expiración del servicio.

- Tabla de Restricciones de Servicios:

Nombre Campo	Descripción
Id_servicio	Identificación del Servicio
Tipo de cliente	Tipos de clientes para los cuales está autorizado el servicio
Id. Materia	Identificación de la Materia

Los datos en las tablas se ingresarán de acuerdo a las definiciones realizadas en la especificación de la aplicación.

En la tabla clientes deberán estar todos los usuarios que tengan algún permiso para usar un servicio, para que Kerberos pueda verificar su autenticidad.

La misma esta compuesta por identificacion del usuario, clave (password del usuario encriptada con la Master Key de Kerberos usando DES) y datos de expiración de la clave.

Es importante destacar que las claves no están en txto plano porque por más que estas tablas estén en un server muy seguro, si algún impostor accede a ellas no encontrará la password del usuario y al no saber cual es la master key de Kerberos serán ínfimas las posibilidades de poder obtenerla.

En la tabla de servicios estarán todos los servicios que puedan llegar a pedir los usuarios y que estén resguardados por Kerberos.

Esta compuesta por la identificación del servicio, clave del servicio y clave de acceso al sevicio también encriptadas con la master key de kerberos por el motivo anteriormente mencionado y el tiemp de expiración de la clave del servicio.

La tabla de restricciones de servicios contiene todas las combinaciones de servicio, tipo de cliente y materia autorizadas.

Esta tabla será consultada por cada server de servicios para determinar si un usuario cualquiera está autorizado a usar un servicio.

Las restricciones estarán dadas por el tipo de usuario.

7.4.7.- Como interactua Kerberos con la aplicación

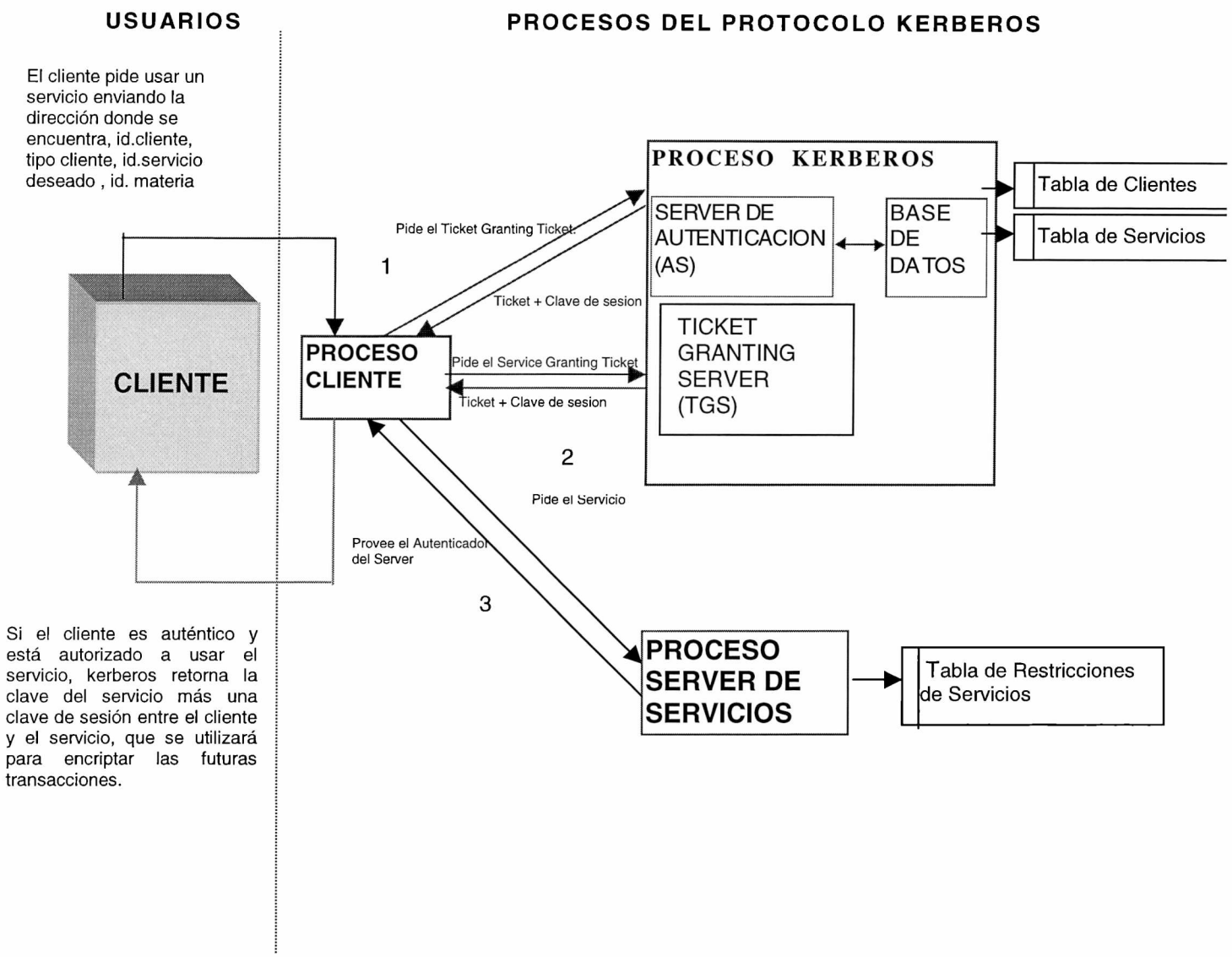
Para poder probar el buen funcionamiento del protocolo Kerberos es necesario contar con una aplicación Kerberizada. Dicha aplicación será implementada en Java y permitirá el acceso remoto de los alumnos de la Facultad de Ciencias Exactas a distintos servicios.

Cuando un cliente quiera utilizar algún servicio deberá primero ingresar su Id. De Usuario y password y luego seleccionará su tipo (Administrativo, Alumno, Profesor, Otros), el servicio deseado y la materia. Por ejemplo:

Id.Usuario: MARIA
Clave: 9481
Tipo Usuario: Alumno
Servicio: Consultar Notas
Id.Materia Algebra I

Una vez que el usuario ingresa los datos especificados anteriormente deberá ser autenticado y autorizado para poder utilizar el servicio, es en ese momento cuando Kerberos interactua con la aplicación kerberizada otorgando seguridad a la red de redes.

Esquema que permite observar con mayor claridad la comunicación entre los clientes y el servidor Kerberos, cuando un cliente quiere acceder a un servicio.



La Tabla de Clientes la lee y escribe solo Kerberos y la utiliza para autenticar a los clientes y obtener sus claves.

La Tabla de Servicios la lee y escribe solo Kerberos y la utiliza para verificar que el servicio exista y obtener sus claves.

La tabla de Restricciones de Servicos la ven y escriben todos los server de servicios y la utilizan para saber si el cliente está autorizado a usar el servicio.

7.4.8.- Refinamiento del Protocolo Kerberos

1- El proceso cliente pide al AS que autentique al cliente.

Cliente _Autentico(Id. Cliente, ADc) ;

- Hay que autenticar al cliente, para ello verifica que la Id. Del Cliente exista en la Tabla de Clientes. Si no existe o su clave expiró retorna que el cliente no es válido. En caso contrario obtiene la clave del cliente

(pasword del cliente encriptada con DES) que usara para encriptar la información que enviará al cliente.

- Kerberos conoce la Id.TGS y su clave.
- Crea una clave de sesión para ser usada entre el cliente y el TGS.
- Arma el Ticket encriptado con la clave del TGS que contendrá los siguientes datos:
    1. Clave de sesión entre el Cliente y el TGS.
    2. Id. Cliente.
    3. ADc (dirección de la computadora donde está el cliente)
    4. Id TGS.
    5. Sello de hora
    6. Tiempo de vida
- Envía la siguiente información al cliente toda encriptada con la clave del cliente para que este la desencripte y Pida el servicio al TGS mediante el ticket que recibe.

Entonces el cliente deberá:

- Desencriptar lo recibido con su clave privada para obtener el Ticket tgs que deberá enviar al TGS.
- Armar un Autenticador del cliente con los siguientes datos: Id. Cliente, ADc y Sello de hora.

2- El proceso cliente pide el servicio deseado al TGS

Pide_Servicio(Id. Servicio Pedido, Ticket tgs, Autenticador cliente)

- El TGS Verifica que el Id. de servicio pedido exista en la Tabla de Servicios. Si no existe retorna que el Servicio deseado no existe. En caso contrario desencripta el Ticket con su clave privada para obtener la Id. Cliente, ADc y la clave de sesión entre el cliente y él.
- Desencripta el Autenticador del cliente con la clave de sesión obtenida del Ticket para obtener el Id. Cliente y la ADc.
- Compara los Id. Clientes y sus direcciones obtenidas del Ticket y del Autenticador, si coinciden quiere decir que el cliente es aquel para el cual se imprimió el Ticket (es auténtico).
- Crea una clave de sesión aleatoria para el cliente y el server de servicios.
- Crea el Ticket de servicio con la siguiente información:
    1- Clave de sesión entre el Cliente y el server de servicio.
    2- Id. Cliente.
    3- ADc (dirección de la computadora donde está el cliente)
    4- Id Servicio
    5- Sello de hora
    6- Tiempo de vida
- Retorna al cliente la siguiente información encriptada con la clave de sesión entre el Cliente y el TGS:

    Clave de sesión entre el cliente y el server de servicio
    Ticket de servicio

Luego el cliente la desencripte con la clave de sesión y Pide el servicio al Server de Servicio mediante el ticket de servicio que recibe.

Entonces el cliente deberá:

- Desencriptar lo recibido con su clave de sesión entre el cliente y el TGS para obtener el Ticket de servicio que deberá enviar al server de servicio y la clave de sesión con la que deberá encriptar el autenticador.
- Armar un Autenticador del cliente encriptado con la clave de sesión entre el cliente y el server de servicios con los siguientes datos: Id. Cliente, ADc y Sello de hora.

3- El proceso cliente pide el servicio deseado al Server de Servicio

Function Pide_ServicioServer(Ticket servicio, Autenticador cliente, Tipo de Cliente, Id. materia) : Booleano

- Desencripta el Autenticador del cliente con la clave de sesión obtenida del Ticket para obtener el Id. Cliente, la ADc. y el ID.Servicio Pedido
- Compara los Id. Clientes y sus direcciones obtenidas del Ticket y del Autenticador, si coinciden quiere decir que el cliente es aquel para el cual se imprimió el Ticket (es auténtico).
- Verifica en la tabla de restricciones de servicios que el servicio pedido sea válido para el tipo de cliente y la Id. materia especificados. Si es válido entonces el cliente está autorizado a usar el servicio y el proceso retornará al cliente la clave de acceso al servicio y dos listas una que contiene el ticket TGS con la clave de sesión y otra que contiene el ticket de servicio con su clave de sesión para poder reusar en futuros pedidos esos tickets, en caso contrario retornará que el cliente no está autorizado para usar el servicio.

La autorización se realiza en cada server de servicio porque el modelo de autorización Kerberos está basado en el principio de que cada servicio conoce que usuarios pueden utilizarlo y que forma de autorización es apropiada.

Las ventajas de este modelo es que todos los servicios tienen una pequeña lista privada de usuarios autorizados.

- Los servicios mantienen su propia lista, no dependiendo continuamente de una network para el servicio de autorización.
- No hay un modelo único aplicable a todos los servicios, un servicio designado, tiene la opción de usar las librerías del modelo de autorización o crear un modelo diferente que podrá adaptarse al servicio en particular.
- La cantidad de información almacenada requerida para la información de autorización es proporcional al número de servicios.

En nuestro caso cada server de servicios consultará la tabla de Restricciones para determinar si el usuario esta autorizado a usar el servicio deseado.

7.5.- Implementación del protocolo Kerberos

7.5.1.- Lenguaje utilizado

Utilizaremos el lenguaje JAVA, usando para su programación la herramienta Visual Age for Java 1.0, los accesos a métodos remotos los realizamos mediante RMI (Invocación de Métodos Remotos) y la base de datos la definimos en Oracle y en SQL Server, usando los drivers correspondientes para el acceso a los datos de sus tablas.

7.5.2.- Definición de Paquetes

Para implementar el Modelo de Autenticación y Autorización Kerberos, debemos contar con la implementación de Kerberos y un Aplet que permita a un usuario cualquiera pedir un servicio.

Por al motivo organizamos la implementacion de la siguiente manera:

Proyecto Nuestro

- Paquete Clientes
- Paquete Cifrado
- Paquete Kerberos

Los tres paquetes están compuestos por varias Clases. Tanto algunas Clases del paquete Kerberos como del paquete Clientes deben importar las clases del paquete Cifrado porque todos los mensajes estan encriptados con el método de encriptación DES implementado en este paquete.

7.5.3.- Clases e Interfaces Principales

Dentro del paquete Kerberos hay una interface:

- **Interface P_Cliente** (parametros de entrada)
  Public String **Seguridad** (parametros de entrada)
  Public String **Registracion** (parametros de entrada)

y una clase que implementa la interface anterior

.

- **Class Proceso_Cliente**(parametros de entrada) **implements P_Cliente**

  Main()
  Public String **Seguridad** (parametros de entrada)
  { implementación.....
  }
  Public String **Registracion** (parametros de entrada)
  { implementación.....
  }

La interface descripta es a la que accederán los usuarios desde los equipos remotos con la utilización de RMI, por lo tanto los métodos que los usuarios usarán desde equipos remotos son Registración y Seguridad.

También dentro del paquete Kerberos hay otras clases que solo pueden ser vistas dentro de este paquete que son utilizadas como bibliotecas por los métos Registración y Seguridad.

7.5.4.- Método registración

El método Registración del paquete Kerberos lo que hace es incorporar nuevos usuarios a la bases de datos Kerberos a la tabla de Clientes, ingresando la identificación del cliente y la pasword del mismo encriptada con Des usando la master Key de Kerberos (que la tomamos como fija), Las claves se guardan encriptadas con la master key de kerberos para que nadie que acceda a la tabla pueda tener acceso directo a la password del cliente.

Este método, primero se fija que no haya ningun usuario en la tabla con esa identificación de usuario, y luego lo agrega a la tabla. Este método es llamado en forma remota por todos los usuarios que quieran registrarse para poder acceder a algún servicio definido para Alumnos, Profesores, Administrativos.

Este método devuelve un String porque en caso que exista un usuario con la identificación ingresada, devuelve un cartel que dice "Ingrese otra identificación de usuario", en caso que pueda ingresarlo como nuevo usuario, devuelve el String en blanco. Cada aplicación de Usuario se encargará de editar el cartel en pantalla.

7.5.5.- Método seguridad

El método Seguridad del paquete Kerberos determina si el usuario que quiere usar un servicio específico esta autenticado y autorizado para hacerlo. Dicho método tomo como datos de entrada la Identificacion del Cliente, la identificación del servicio, la direccion de la computadora de donde hace el pedido, el tipo de cliente (Alumno, Profesor, Administrativo), la materia, el ticket TGS anterior, el Ticket de servicio anterior, los últimos en caso que su tiempo de vida no haya expirado.

Entonces cada usuario antes de poder usar un servicio deberá llamar al método remoto Seguridad.

En caso que el usuario este autorizado a usar el servicio el método devuelve un String, encriptado con la clave privada del usuario. Al desencriptar dicho string, nos encontramos con la clave de sesión entre el cliente y el server de servicios, la lista TGS que contiene al ticket TGS y la lista de servicios que contiene al ticket de servicios., la clave de acceso al servicio y un cartel en blanco.

Si el Usuario no esta autenticado o autorizado, devuelve un string encriptado con la clave privada del usuario con la clave de sesion en blanco, la lista tgs en blanco, la lista de servicio en blanco, la clave de servicio en blanco y el cartel con un mensaje que explica porque el usuario no puede acceder al servicio.

En cualquiera de los dos casos anteriores las aplicaciones de usuarios desencriptan el string recibido y tienen métodos dentro de la clase resultados que toman del String lo que es necesario por ejemplo para obtener la clave de acceso al servicio, obtener el cartel, etc..

También cada aplicación de usuario sera la encargada de mostrar en pantalla el cartel con la explicación de porque no pueden usar el servicio.

Por lo tanto estas aplicaciones de usuarios deben ser aplicaciones Kerberizadas.

Desidimos hacer un único método Seguridad que interactue con cada aplicación remota para que los diseñadores de dichas aplicaciones no tuvieran que estar tan

empapados en el manejo de Kerberos. Seguridad es la que hace todo este manejo, invocando a métodos del paquete Kerberos.

Al método Seguridad ingresan los parámetros y este comienza a hacer las operaciones necesarias para imprimir el Ticket TGS y el Ticket de Servicios y obtener la clave de acceso al servicio.

- Descripción de lo que hace el método Seguridad
    1. De entrada recibe los siguientes parámetros:
        - Identificación del Cliente
        - Direccion del Cliente
        - Identificación de Materia
        - Identificación de Servicio
        - Lista TGS encriptda con la clave privada del Cliente
        - Lista de Servicio encriptada con la clave de sesion contenida en la lista TGS

    2. Verifica que el Ticket TGS anterior recibido como parámetro sea distinto de blanco o su tiempo no haya expirado.

    3. En caso que el Ticket TGS anterior sea blanco o su tiempo haya expirado, genera un nuevo ticket TGS para el cliente, para ello llama al método AS con los siguientes parámetros de entrada :
        - Identificación del Cliente
        - Direccion del Cliente
        - Identificación del TGS (que es fija)
        - TimeStamp

    4. El AS verifica que el cliente sea Autentico (que exista en la tabla de clientes de la Base de Datos Kerberos) , luego emite un Ticket para acceder al TGS y devuelve una lista encriptada con la clave privada del cliente que contiene los siguientes datos:
        - Clave de sesión entre el Cliente y el TGS
        - Identificación del TGS
        - TimeStamp
        - Tiempo de vida
        - Ticket TGS encriptado con la clave privada del TGS

    5. Una vez que el AS devuelve la lista, Seguridad desencripta la lista recibida con la clave privada del cliente , verifica que esa lista sea distinta de "no puede" eso quiere decir que el cliente es auténtico y toma el Ticket TGS, la clave de sesión que usará para los futuros intercambios entre el Cliente y el TGS y arma un autenticador encriptado con la clave de sesión mencionada, que asegura que el ticket para el cual fue impreso el ticket y aquel que generó el autenticador es el mismo.

    6. Llama al método TGS con los siguientes parámetros de entrada:
        - Identificación del Servicio
        - Identificación del Cliente
        - Dirección del Cliente
        - Ticket TGS
        - Autenticador
        - Identificador del TGS
        - Lista de Servicios anteriror

7. El TGS verifica la existencia del servicio pedido, desencripta el Ticket TGS con la clave privada del TGS, obtiene la clave de sesión que usará para desencriptar el autenticador, compara que el Identificador de usuario y la dirección de red del Ticket y el Autenticador coincidan, si es así quiere decir que el cliente es aquel para el que se imprimió el ticket, genera el Ticket de servicio y devuelve una lista encriptada con la clave de sesión que contiene los siguientes datos:

- Clave de sesión a ser usada por el cliente y el server de servicio
- Identificación del Servicio
- TimeStamp
- Ticket de Servicio encriptado con la clave privada del servicio, para que solo él pueda desencriptarlo en caso contrario, no le genera el ticket de servicio.

8. Luego Seguridad desencripta con la clave de sesión la lista recibida por el TGS y obtiene la nueva clave de sesión para usar con el server de servicio, el Ticket de Servicio y genera un nuevo autenticador encriptado con esta nueva clave de sesión. Luego llama al método ServerServicios con los siguientes parámetros de entrada:

- Ticket de Servicio
- Autenticador
- Identificación Servicio
- Tipo Cliente
- Identificación de Materia
- Identificación del Cliente

9. El ServerServicio desencripta el Ticket de Servicio con la clave privada del Servicio, obtiene la clave de sesión que usará para desencriptar el autenticador, compara que el Identificador de usuario y la dirección de red del Ticket y el Autenticador coincidan, si es así quiere decir que el cliente es aquel para el que se imprimió el ticket, y devuelve la clave de acceso al servico pedido, dicha clave la devuelve encriptada con la clave de sesion entre el cliente y el server de servicio.

10. Luego Seguridad desencripta con la clave de sesión la lista recibida y obtiene la clave de acceso al servicio para armar la lista que devolverá al usuario encriptada con la clave privada del usuario. Dicha lista contendrá:

- Clave de acceso al servicio
- Lista TGS encriptada con la clave del cliente
- Lista del Servicio encriptada con la clave de sesión que está en la laista TGS.
- Mensaje que estará en blanco si el usuario puede usar el servicio o especificará porque el usuario no pudo acceder al servicio , por ejemplo por no estar autorizado.
- Nota: las listas tanto la del TGS y la de Servicio estarán entre corchetes.

Cuando el Ticket TGS o el Ticket de servicio pueden ser blancos?, cuando el usuario no esta autenticado (no existe en la tabla de Clientes) o no está autorizado a usar el servicio (la combinación tipo de usuario, tipo de servicio e identificación de materia

no existe en la tabla de restricciones de servicio) y cuando el servicio pedido no existe en la tabla de Servicios.

7.5.6.- Otras clases utilizadas

Dentro del paquete Kerberos hay otras clases que contienen varios métodos que se usan en los dos métodos anterirores, dichas clases son las siguientes:

- **Class Autenticador**

Esta clase está compuesta por los siguientes métodos:

- String **crearAutenticador** (String id, String dir)
  Retorna el autenticador de acuerdo al identificador de usuario y la dirección de la computadora donde está el usuario ingresados como parámetros, dicho autenticador se compone de la siguiente manera:
  Autenticador = id_cliente +" "+ dir_cliente+" "+ sello de hora;

- String **obtenerDireccion** (String aut)
  Retorna la dirección que está en el Autenticador para poder manipularla.

- String **obtenerIdCliente** (String aut)
  Retorna la identificación del cliente que se encuentra en el Autenticador para poder manipularla.

- String **obtenerSelloHora** (String aut)
  Retorna el sello de hora impreso en el Autenticador para poder manipularlo.

- **Class Cliente**

Esta clase está compuesta por los siguientes métodos:

- boolean **autenticoCli**(String id_cli)
  Retorna true si el cliente pasado como parámetro existe en la tabla de Clientes;

- String **clavePrivada** (String id_cli)
  Retorna la clave privada del cliente pasado como parámetro, para poder encriptar con dicha clave los mensajes pasados al cliente.

- boolean **igualCliente**(String id_cli1, String id_cli2)
  Retorna true si los clientes pasados como parámetros coinciden.

- boolean **igualDireccion**(String dir1, String dir2)
  Retorna true si las direcciones pasados como parámetros coinciden.

- **Class Genkerb**

  Esta clase tiene varios métodos que permiten verificar la autenticidad del cliente y armar la lista que contiene el Ticket TGS que va del AS al Cliente

encriptada con la clave privada del cliente (método AS), Obtener el identificador y la clave privada del TGS, Obtener la clave de sesión que es una clave que se genera en forma aleatoria, obtener el Autenticador, el Ticket de Servicio y el Ticket TGS encriptados con sus respectivas claves, Armar la lista que contiene el ticket de Servicio y Verificar que el servicio pedido exista y devolver la clave de acceso al servicio encriptada con la clave de sesion entre el cliente y el server de servicio. Dichos métodos son:

- String **as**(String id_cli,String dir, String ide_TGS, Timestamp sello)
  Este es uno de los métodos principales de esta implementación. Retorna la lista que contiene el ticket tgs que el AS devuelve al cliente encriptada con la clave del cliente, verificando previamente que el cliente sea válido y obteniendo el ticket tgs.

- boolean **autenticoTgs** (String idtgs)
  Retorna true si el identificador del TGS ingresado como prámetro es válido. En esta implementación usamos un identificador del TGS fijo que es "TGS".

- String **clavePriTgs**(String idtgs)
  Retorna la clave privada del TGS ingresado como prámetro. Para poder encriptar con dicha clave el ticket enviado al TGS.

- String **claveSesion**()
  Retorna la clave de sesión para los intercambios entre el cliente y el TGS y el cliente y el server de servicios. Dicha clave es aleatoria.

- String **obtenerAutenticador**(String id_cliente, String dir, String clave_sesion)
  Retorna el autenticador del cliente ingresado como parámetro encriptado con la clave de sesión también pasada como parámetro.

- String **obtenerTicketServicio**(String id_servicio, String dir_cli, String id_cli, String cla_se,Timestamp tis)
  Retorna el ticket obtenido para el server de servicios encriptado con la clave del servicio.

- String **obtenerTicketTGS**(String id_cliente, String dir_cli, String ide_tgs, String cla_se,Timestamp ts)
  Retorna el ticket obtenido para el TGS encriptado con la clave del TGS.

- String **sacarClaveSesion**(String lista)
  Retorna la clave de sesión de la lista ingresada como parámetro.

- String **sacarTicketServicio**(String lista)
  Retorna el ticket de servicio de la lista ingresada como parámetro.

- String **sacarTicketTgs**(String lista)

Retorna el ticket TGS de la lista ingresada como parámetro.

- String **sellodehora**(String lista)
  Retorna el sello de hora de la lista ingresada como parámetro.

- String **serverServicios**(String ticket, String auten, String id_serv, String tipo_cli, String id_mat, String id_cli)

  Este es otro de los métodos más importantes de esta clase. Retorna la clave de acceso al servicio encriptada con la clave de sesión entre el cliente y el server de servicios, verificando previamente que los datos que están en el ticket de servicio y el autenticador del cliente coincidan y que el cliente ingresado este autorizado a usar el servicio pedido. Si alguna de las condiciones no se cumplen devuelve un mensaje explicando porque el cliente no puede usar el servicio.

- String **serviPed**(String lista)
  Retorna el servicio pedido de la lista ingresada como parámetro.

- String **tgs**(String id_serv, String id_cli, String ad_cli, String tick_tgs, String auten, String id_tgs, String listaServi)

  Este es otro de los métodos principales de esta implementación. Retorna la lista que contiene el ticket de servicio que el TGS devuelve al cliente encriptada con la clave de sesión entre el cliente y el TGS, verificando previamente que el servicio pedido exista y que los datos del ticket TGS y el autenticador del cliente coincidan. Si alguno de las premisas no coinciden devuelve el mensaje correspondiente explicando porque el usuario no puede usar el servicio pedido.

- String **tiempoVida**(String lista)
  Retorna el tiempo de vida de la lista ingresada como parámetro.

- **Class Servicios**

  Cuenta con métodos que permiten verificar la existencia del servicio, obtener la clave de acceso al servicio, obtener la clave privada del servicio y verificar que el cliente este autorizado a usar el servicio.

  - String **claveAccesoServicio**(String id_serv)
    Retorna la clave de acceso al servicio ingresado como parámetro.

  - String **clavePrivadaServicio**(String id_serv)
    Retorna la clave privada del servicio ingresado como parámetro.

  - boolean **verificarAutorizacion**(String id_servicio, String tip_cli, String tip_mat)

Retorna true si el tipo de cliente y materia ingresados omo parámetros están autorizados a usar el servicio también ingresado como parámetro.

- boolean **verificarExistencia**(String id_servicio)
  Retorna true si el servicio ingresado como parámetro existe en la tabla de servicios.

- **Class Ticket**

Esta compuesta por métodos que permiten crear los ticket TGS y de Servicio y obtener los elementos que hay dentro de cada uno de ellos.

- String **obtenerClaveSesion**(String tick)
  Retorna la clave de sesión del ticket ingresado como parámetro.

- String **obtenerCuartoElemTick** (String tick)
  Retorna la clave de sesión del ticket ingresado como parámetro.

- String **obtenerDireccion**(String tick)
  Retorna la dirección del ticket ingresado como parámetro.

- String **obtenerIdCliente** (String tick)
  Retorna el cliente del ticket ingresado como parámetro.

- String **obtenerSelloHora** (String tick)
  Retorna el sello de hora del ticket ingresado como parámetro.

- String **obtenerTiempo** (String tick)
  Retorna el tiempo de vida del ticket ingresado como parámetro.

- String **ticketServicio**(String id_cli, String dir_cli, String id_servicio, String clave_sesion,Timestamp tis)
  Retorna el ticket de servicio.

- String **ticketTGS** (String id_cliente, String dir_cliente,String clave_sesion, String ide_TGS,Timestamp tis)
  Retorna el ticket TGS.

- **Class Tiempo_Ticket**

Permite verificar que el tiempo de los ticket no haya expirado.

- boolean **esvalido** (String hora, Timestamp tis) {
  Este método retorna verdadero cuando el sello de hora de la máquina es anterior al tiempo hasta el cual es válido el ticket

- String **tiempoHasta**(Time ti, Timestamp tis)
  Este método suma el time stamp en el que el ticket fue generado

mas el tiempo de vida del mismo para sacar su máximo tiempo de vida

Para poder demostrar la utilización de Kerberos implementamos una aplicación muy sencilla, ya que ese no era el objetivo de nuestra tesis, que permite ingrasar los parámetros necesarios e invocar a los dos métodos remotos de Kerberos presionando el botón correspondiente, dichos métodos han sido explicados anteriormente y son Registración y Seguridad. Luego la aplicación muestra un mensaje devuelto por los procesos de Kerberos con la clave de acceso al servicio o con el motivo por el cual el usuario no puede acceder al servicio pedido.

7.6.- Implementación de JDBC - ODBC

En nuestra implementación del Protocolo de Autenticación y Autorización Kerberos hay tres clases que cuentan con métodos que acceden a las tablas de la Base de Datos Kerberos. Este acceso lo realizamos a través del paquete JDBC, mencionado en el capítulo 5 de JDBC de la tesis, que nos permite enviar consultas SQL a una Base de Datos y recibir el resultado de dichas consultas.

Las pruebas las realizamos utilizando dos Bases de Datos distintas, Oracle 7 y SQL*Server, las cuales utilizan distintos tipos de drivers.

En el lugar que realizamos el desarrollo del proyecto tenemos instalada la Base de Datos Oracle 7 con su driver correspondiente y en el laboratorio LINTI de la Facultad de Ciencias Exactas de nuestra ciudad, realizamos pruebas con la Base de Datos SQL*Server mediante el driver ODBC.

Mediante la conexión y uso de las dos Bases de Datos y sus drivers correspondientes demostramos que el proyecto de Autenticación y Autorización Kerberos implementado en Java se puede conectar a distintas Bases de Datos, teniendo en cuenta que solo hace falta que esté presente el driver correspondiente para poder realizar el acceso.

Como ya especificamos en el capítulo 5 de JDBC, el uso de JDBC requiere de las siguientes acciones:

- Cargar los Drivers correspondientes.
- Indicar qué Base de Datos se usará.
- Conectarse a la Base de Datos seleccionada.
- Enviar sentencias SQL a través de la conexión.
- Obtener la respuesta deseada.

Podemos detallar aún más a través de los siguientes pasos:

1. Registrar e instalar el Driver.
2. Crear la conexión usando la clase Connection.
3. Usar métodos de la API para interactuar con la fuente de datos, utilizando la clase Statement.
4. Obtener los resultados usando métodos de la API en la clase ResultSet.

Explicamos los pasos detallados anteriormente con las Bases de Datos utilizadas para las pruebas.

7.6.1.- Base de Datos Oracle 7

Oracle provee dos categorías de drivers *JDBC THIN* para utilizar en applet y aplicaciones y *JDBC OCI* solo para aplicaciones.

JDBC THIN no depende de SQL*Net ni de nada.

JDBC OCI utiliza llamadas que entran a la Base de Datos a través de SQL*Net.

JDBC OCI provee una implementación de interface JDBC que usa OCI (Oracle Call Interface) para interactuar con la Base de Datos Oracle. Este driver posee llamadas nativas (escritas en C), por ende, es específico para cada plataforma. Se utiliza software cliente Oracle 7.3.4 incluyendo SQL*Net.

Las aplicaciones deberán seguir una serie de pasos para usar JDBC OCI:

- Importar las clases JDBC

```
import java.sql.*;
```

- Registrar el driver OCI

```
DriverManager.registerDriver
(new oracle.jdbc.driver.OracleDriver());
```

- Abrir la conexión a la base

```
Connection conn =
DriverManager.getConnection("jdbc:oracle:oci7://@mi_base", "usuario",
"clave" );
```

El método getConnection devuelve un objeto de la clase Oracle Connection. Nos conectamos al protocolo jdbc, que a su vez tiene un subprotocolo Oracle (en URL los subprotocolos se separan con :), y hay otro subprotocolo que es oci7. @mi_base es el nombre por el cual entramos al SQL*Net (TSNames para referenciar la base)

JDBC THIN utiliza sockets de Java para comunicarse directamente con la Base de Datos Oracle. Este provee su propia implementación en TCP/IP de SQL*Net. Este driver está escrito enteramente en Java y por este motivo es independiente de la plataforma.

No requiere software cliente de Oracle pero debe existir un *listener TCP/IP* del lado del servidor.

Las aplicaciones deberán seguir una serie de pasos para usar *JDBC THIN*:

- Importar las clases JDBC

```
Import java.sql.*;
```

- Registrar el driver THIN

```
DriverManager.registerDriver
```

```
(new oracle.jdbc.driver.OracleDriver());
```

- Abrir la conexión a la base

```
Connection conn = DriverManager.getConnection
("jdbc:oracle:thin://@mi_host:1521:orcl", "usuario", "clave" );
```

En nuestra implementación elegimos usar el driver de oracle *JDBC THIN* porque puede usarse tanto para applets como para aplicaciones, además al ser un driver de tipo Native Protocol pure Java Driver, no debe estar del lado del usuario, sólo en el server, explicado en el capítulo 5 de la tesis.

El uso de JDBC con Oracle lo realizamos mediante las siguientes sentencias:

1. Registrar e instanciar el Driver

```
Class.forName
("oracle.jdbc.driver.OracleDriver").newInstance();
```

2. Crear la Conexión

   El string *url* (Uniform Resource Locator) da información para localizar un recurso en Internet, considerada como una dirección.
   La sintaxis standard para los JDBC URL:

```
jdbc:< subprotocol >:< subname >
```

   La primer parte específica el protocolo usado para acceder a la información , seguido siempre por dos puntos, el resto especifica la localización de la fuente de datos, el protocolo *JDBC url* es siempre *jdbc*, el *subprotocolo* es el nombre del Driver o el nombre del mecanismo de conectividad a la Base de Datos, el *Subname* es la manera de identificar la Base de Datos.

```
String dir = "127.0.0.1" (Dirección IP, si la Base de Datos esta en una
máquina que no está en red, en caso contrario estará la dirección de red
IP correspondiente );
String puerto = "1521"  (Puerto utilizado para comunicarse con la Base de
Datos)
String url = "jdbc:oracle:thin:@dir:puerto:ORCL";
Connection con =  DriverManager.getConnection(url,"kerb","kerb");
                                                    |       |
                                                 Usuario   Clave
```

3. Usar métodos de la API para interactuar con la fuente de datos usando la clase Statement

```
Statement stmt = con.createStatement();
```

4. Obtener los resultados usando métodos de la API en la clase ResultSet

```
ResultSet rs = stmt.executeQuery("select * from clientes where '"+ cli
+"'=id_cliente");
if (rs.next() ) /* encontró algún elemento en la tabla que cumpla con la
condición*/
      { acción a realizar }
```

5. Cerrar el conjunto de resultados, las sentencias y la Conexión

```
rs.close();
rtmt.close();
con.close();
```

7.6.2- Base de Datos SQL*Server

El driver usado para conectarse a la Base de Datos es de tipo Bridge ODBC-JDBC descripto en el capítulo 5 de JDBC, que provee acceso JDBC a través de drivers ODBC.

Los pasos para usar JDBC-ODBC con SQL*Server son los siguientes:

1. Registrar e instanciar el Driver

```
Class.forName
("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
```

2. Crear la Conexión

   La sintaxis standard para los JDBC URL:

```
jdbc:< subprotocol> :< subname >
```

   El *subprotocolo ODBC* es un caso especial ha sido reservado para URLs que especifican nombres de fuentes de datos ODBC-style y tiene la característica especial de permitir cualquier número de valores atributos a ser especificados después del *subname*.

```
String url = "jdbc:odbc:kerberos";
Connection con = DriverManager.getConnection(url,"sa","");
```

3. Usar métodos de la API para interactuar con la fuente de datos usando la clase Statement

```
Statement stmt = con.createStatement();
```

4. Obtener los resultados usando métodos de la API en la clase ResultSet

```
ResultSet rs1 = stmt1.executeQuery("insert into clientes VALUES
('"+cliente+"','"+cla_cli+"', NULL)");
```

5. Cerrar el conjunto de resultados, las sentencias y la Conexión

```
rs.close();
stmt.close();
con.close();
```

Los accesos a la base Kerberos, en nuestro proyecto los realiza el server en forma local, porque para resguardar la información almacenada en las tablas de esta Base de Datos los usuarios no pueden tener acceso a las mismas.

Los usuarios se comunican con el server Kerberos, solicitan la acción necesaria, por ejemplo, determinar si un cliente es válido, para ello el server accede a las tablas de la Base de Datos Kerberos en forma local y envía el resultado al usuario.

Por esto no es necesario para esta implementación que los usuarios configuren el driver JDBC-ODBC en sus máquinas. Esta configuración solo se hará en el server.

7.7.- Implementación de R.M.I. para accesos remotos

Pensamos que una implementación posible para comunicar el cliente con el servidor era RMI, protocolo cliente/servidor basado en el lenguaje Java. Otra alternativa era *Sockets* que son flexibles y suficientes para una comunicación general y soporta el lenguaje Java. Sin embargo los *sockets* requieren que el cliente y el servidor se relacionen en protocolos de niveles de aplicaciones para codificar y decodificar intercambio de mensajes, el diseño de tales protocolos es problemático y tiende a error.

Ya que usando JDBC podemos crear aplicaciones que son capaces de acceder a una Base de Datos tanto local como remota y en nuestra implementación debemos acceder a métodos remotos, decidimos utilizar RMI porque se adapta bien al lenguaje Java y nos permite crear aplicaciones distribuídas en forma eficiente, con un diseño más simple y natural.

Como explicamos en el capitulo 6, RMI nos permite llamar a un objeto Java remoto como si este estuviera en la máquina local.

En nuestra tesis esto de gran utilidad porque todos los usuarios que quieran utilizar un servicio se deben autenticar con el Servidor Kerberos remoto llamando al método Seguridad o registrándose como nuevo usuario llamando al método remoto Registración.

7.7.1.- Proceso de desarrollo para una aplicación RMI

Consiste de los siguientes pasos:

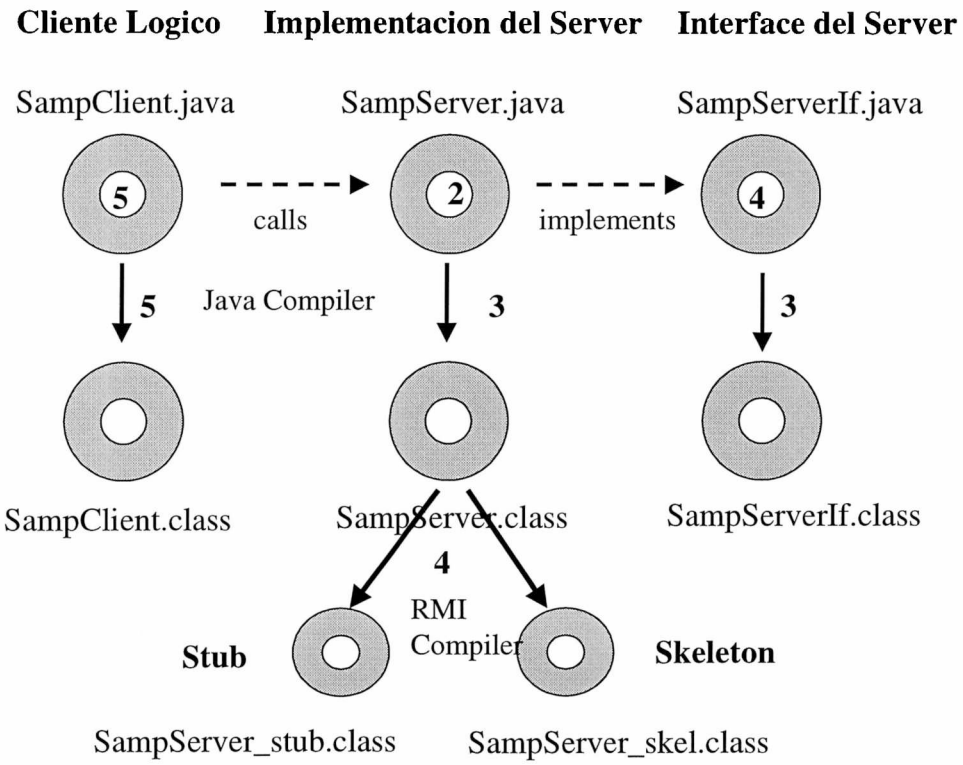
1. Definir la interface pública del server
2. Escribir la implementación del server
3. Compilar la interface del server y el server
4. Ejecutar el compilador RMI para crear las clases *stub* y *skeleton*
5. Escribir y compilar el cliente lógico

La interface pública del server define los métodos que pueden ser invocados desde el cliente.

La clase server implementa la interface pública del server. El código RMI especial es necesario para levantar manejador de seguridad y registrar el server con el RMI registry

El *RMI compiler* genera las clases stub y skeleton desde la clase server.

El cliente lógico usa código RMI especial para observar el nombre del servidor y tomar una referencia al objeto server. Este puede luego usar los métodos públicos del server a través del objeto local



Proceso de Desarrollo RMI

Para utilizar RMI hemos implementamos en el server y en el cliente los siguientes pasos:

1- Del lado del server (paquete Kerberos)

- Definimos la Interface que contienen los métodos que los usuarios accederán en forma remota.

```
import java.rmi.*;
public interface P_Cliente extends Remote
public String Registracion (String id_cliente,String cl_cliente) throws
RemoteException;
public String Seguridad(String id_cliente, String dir_cliente, String
tipo_cliente,String id_materia, String id_servicio,String listTgs, String
listServ) throws RemoteException;
```

En este desarrollo los únicos métodos que podrán accederse en forma remota son:
- *Registración* ,que permite a cada usuario registrarse en la Base de Datos Kerberos, y
- *Seguridad,* que autentica y autoriza a los usuarios a utilizar un servicio determinado

Todas las interfaces que definimos para RMI extienden de la clase Remote.

- Implementamos los métodos remotos del servidor, o sea, todos los métodos que fueron declarados en la interface remota.

```
import java.rmi.*;
import java.rmi.server.*;
public class ProcesoCliente extends UnicastRemoteObject implements
P_Cliente {
public static void main(String args[])throws RemoteException {
        ProcesoCliente procli=null;
        try
        {
        System.setSecurityManager(new RMISecurityManager());
        System.out.println ("Levantando el manejador de Seguridad");
        procli = new ProcesoCliente();
        Naming.rebind("rmi:///Servidor",(Remote)procli);
        System.out.println("Seguridad server is ready");
        }
        catch (Exception e)
        {
                System.out.println("Excepción " +e+ e.getMessage());
        }
        return;
}

public String Registracion(String cliente, String cl_cliente) throws
RemoteException {
                ..............
        Implementación del método Registración
          ..........
         }

public String Seguridad(String id_cliente, String dir_cliente, String
tipo_cliente,String id_materia, String id_servicio,String listTgs, String
listServ) throws RemoteException {
          ..............
        Implementación del método Seguridad
          ..........
         }
```

Primero levantamos el manejador de seguridad y luego con el Naming.rebind registramos al servidor Kerberos con el nombre Servidor.

- Creamos el Stub y el Skeleton ejecutando:

  Luego generamos las clases *stub* y el *skeleton* que necesitamos para la comunicación. Es invocado usando el comando ***rmic***

  ```
  rmic kerberos.ProcesoCliente
  ```

  Se generaron un Stub.ProcesoCliente.class y un Skeleton.ProcesoCliente.class

- El server debe registrarse mediante la sentencia *rmiregistry 1099* para que el cliente puede acceder a los métodos remotos del cliente.
  El RMI Registry es un programa que provee los servicios de naming lookup en tiempo de ejecución. Tiene que estar ejecutándose antes que el objeto servidor RMI pueda ser instanciado. Un objeto server tiene que registrarse a si mismo con el RMI registry para ser accesible desde los clientes sobre un puerto por el cual los clientes se comunicaran con el Server, en este nuestro es el puerto 1099.

2- Del lado de los Usuarios (paquete Clientes)

Para llamar a los métodos remotos a través de RMI debemos antes registrar el manejador de seguridad y llamar a los métodos a través de una variable de instancia miCliente, que la inicializamos en el método init() de la siguiente forma:

```
String urlString= "rmi://urano/1099/Servidor";
MiCliente=(P_Cliente)Naming.lookup(urlString);
```

Declaramos la variable *urlstring* que consiste de: protocolo RMI, host al que queremos conectarnos, número de port, nombre definido del lado del server. Por último , vemos a la interface remota P_Cliente usando el Naming.lookup a través del *urlstring* declarado y eso se lo asignamos a la variable de instancia miCliente.
Cada vez que tenemos que llamar a algún método remoto declarado en la interface remota lo hacemos a través de la variable *miCliente*, como si se corriera en forma local.
El llamado a los procesos remotos lo hacemos en el método *action* de la clase *Usuarios* que esta dentro del paquete *Clientes*.

```
resu=miCliente.Seguridad( idusuar,dirusuar,ticl,mat,serped,tgs,ser);
```

7.7.2.- Secuencia de operaciones para ejecutar una aplicación RMI

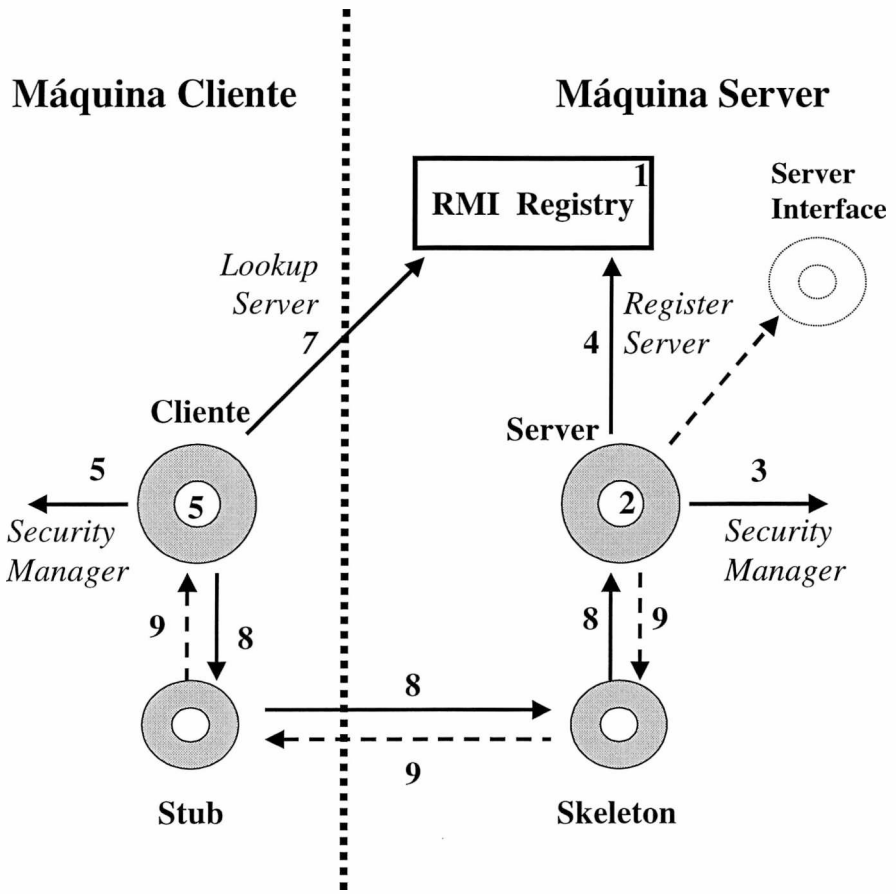
El siguiente gráfico muestra la secuencia de operaciones para ejecutar una aplicación RMI:

Ejecutar el RMI registry sobre la máquina server (1).

Ejecutar la aplicación del server (2), el cual levanta un manejador de seguridad (3) y se registra con el RMI registry (4).

Ejecutar la aplicación del cliente o applet (5), el cual levanta un manejador de seguridad (6) y observa el server usando el RMI registry sobre la máquina remota.

El cliente puede ahora usar un objeto local para invocar métodos a través del stub y skeleton sobre el objeto server (8) . Los objetos resueltos de tales llamados son ruteados al objeto cliente (9).



Ambiente de Ejecución RMI

Las pruebas con RMI las realizamos en el laboratorio LINTI de la Facultad de Ciencias Exactas de esta ciudad.

El server es una de las máquinas llamada URANO y ahí corre nuestro ProcesoCliente y desde otra máquina probamos que un cliente acceda al server.

Los pasos para realizar estas pruebas son los siguientes:

- En el Server

  1. En el path y en el classpath poner el camino necesario para que el servidor Kerberos pueda usar todos los paquetes Java.
  2. Copiar tolos los *.class que componen nuestra implementación Kerberos y compilarlos, ejecutando *javac *.java.*
  3. Crear el Stub y el Skeleton de la clase ejecutable ProcesoCliente.
  4. En otra sección de DOS correr *rmiregistry 1099* que es el puerto por el que se comunica RMI.
  5. Correr el server ejecutando la clase ProcesoCliente, mediante la sentencia *java ProcesoCliente*

- En el Cliente
  1. Ejecutar el Applet que llama a los métodos remotos a través del puerto 1099 utilizando RMI.

7.8.- Implementación del Método DES

DES es el método de encriptación que utilizamos en el Modelo de Autenticación y Autorización Kerberos versión 4, para encriptar tickets y claves (explicado en el capítulo 2).

Implementamos el algoritmo de encriptación DES en el lenguaje de programación Java utilizado para el desarrollo de la tesis.

Para el desarrollo de la misma, creamos un paquete Encriptacion con clases que implementan el algoritmo de encriptación DES, dichas clases se componen de métodos, funciones y arreglos necesarios para dicha implementación.

Las clases son:

- Class Crypt
- Class Cipher
- Class DES
- Class DEA
- Class CipherOutputStream
- Class CipherInputStream
- Class Cifrado

A continuación detallamos los métodos de dichas clases.

7.8.1.- Class Crypt

La clase *Crypt* es simplemente un repositorio de funciones útiles.

La encriptación tiende a requerir varias funciones de transformaciones que convierten arreglos de bytes en arreglos de enteros , y así siguiendo . Esta clase provee un conjunto útil de tales métodos.

Esta clase provee una variedad de métodos útiles para manipulación de arreglos de bytes. Los algoritmos de encriptación casi invariablemente operan sobre arreglos de datos, y estos métodos serán bastante usados frecuentemente para convertir bytes a enteros, de long a bytes, y así siguiendo.

```
Public class Crypt {
// public static final boolean equals (byte[] a, byte[] b)
// public static final void zero (byte[] a, int ao, int l)
// public static final void fill (byte a, byte[] b, int bo, int l)
// public static final void intToBytes (int a, byte[] b, int bo)
// public static final void intsToBytes (byte[] a, int ao, int l, byte[] b, int bo)
// public static final int bytesToInt (byte[] a, int ao)
// public static final void bytesToInts (byte[] a, int ao, int[] b, int bo, int l)
// public static final void longToBytes (long a, byte[] b, int bo)
// public static final long bytesToLong (byte[] a, int ao)
// public static final String bytesToHex (byte[] a)
// public static final String longToHex (long a)
```

Todos los métodos son *static* y *final* .

Static significa que son métodos de clases y no necesitan crear una instancia de la clase *Crypt* para hacer uso de ella.

Declarándolas como *final* le permite a un compilador llevar a cabo algunas optimizaciones. Esto garantiza que ninguna subclase puede sobreescribir estos métodos. Estos pueden ser algunas veces optimizados y alineados. Técnicamente, todos los métodos *static* son *final.* Sin embargo, las declaraciones explícitas sirven como propósito de claridad

El método *equals* determina si los arreglos de dos bytes a y b son iguales. Deben tener la misma longitud y el mismo contenido para ser iguales.

El metodo *zero* completa con l cantidad de ceros el arreglo a, comenzando con un *offset ao.*

El método *fill* completa l bytes del arreglo b con el valor a, comenzando desde *bo*

El método *intToBytes* escribe el entero a como su componente cuatro en un arreglo *b* de cuatro componentes de bytes, comenzando con *bo.* El entero es escrito como el primer byte alto

El método *intsToBytes* escribe l enteros desde un arreglo a como bytes en una arreglo de bytes *b.* Los enteros son tomados desde *offset ao* del arreglo a escritos desde *offset bo* en el arreglo *b.*

El método *bytesToInt* toma cuatro bytes del arreglo a de offset ao y retorna el correspondiente entero

El método byte*sToInts* realiza la operación inversa del método intsToByte(), leyendo l enteros desde el arreglo de bytes a comenzando en el offset ao , escribiéndolos en una arreglo de enteros b comenzando en el offset bo

El método *longToBytes* escribe el long *a* como 8 bytes de arreglo *b*, comenzando en el *offset bo*

El método *bytesToLong* lee bytes desde el arreglo a, comenzando en el offset ao , y retorna el correspondiente long

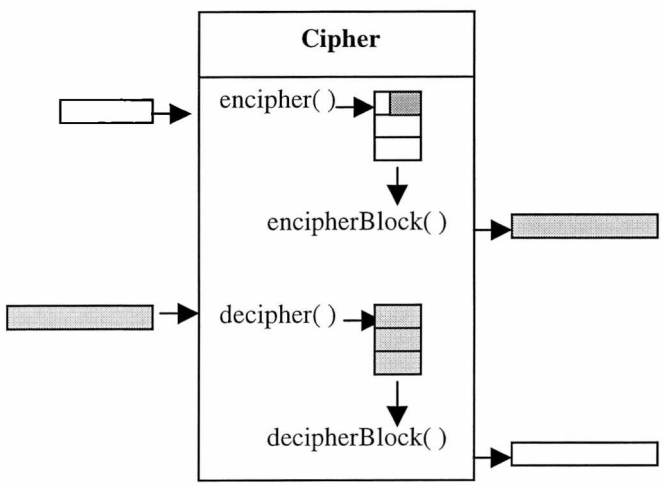
El método *bytesToHex* convierte un arreglo de bytes en un String hexadecimal. El resultado consistiría de dos dígitos hexadecimales para todo byte de entrada.

El método *longtoHex* convierte un long en un String hexadecimal de 16 dígitos long.

7.8.2.- Class Cipher

Un *cipher*, en este caso, es un algoritmo de encriptación. Desarrollamos la implementación de DES.

Esta es la superclase genérica de todos los ciphers. Declara el método básico para encriptar y desencriptar datos. Esta clase es orientada hacia los ciphers que encriptan datos en bloques, y provee métodos que encriptan grandes cantidades de datos haciendo llamados repetitivos a métodos de encriptación de simples bloques realizando repetidas llamadas a métodos de encriptación de bloque simple



Class Cipher

```
public abstract  class Cipher {
// public byte encipher (byte[] plain, int off, int len)...
// public byte encipher (byte[] plain)...
// public byte decipher (byte[] cipher, int off, int len)...
// public byte decipher (byte[] cipher)...
// public abstract int blockSize ( );
// public abstract void encipherBlock (byte[] plain, int po, byte[] cipher, int
co);
```

```
        // public abstract void decipherBlock (byte[] cipher, int co, byte[] plain, int
po);
        }
```

La interface básica a esta clase es a través de los métodos *encipher()* y *decipher()* que encriptan y desencriptan bloques de datos, y el método *blockSize()* que retorna el tamaño del bloque operado por el algoritmo de encriptación. Las implementaciones de los métodos *encipher()* y *decipher()* hacen uso de los métodos *encipherBlock()* y *decipherBlock()* para encriptar datos en unidades del tamaño del bloque de encriptación.

Las actuales implelentaciones de esta clase deben proveer implementaciones de los métodos *blockSize(), encipherBlock()* y *decipherBlock().*

El método encipher encripta len bytes del arreglo plain comenzando con el *indice off* y retorna el resultado encriptado.

Primero creamos un buffer cipher para mantener los datos encriptados resultantes. Este buffer debe ser grande suficientemente para mantener detecto plano y debe también ser múltiplo del tamaño del bloque encriptado. Luego encriptamos el texto plano en pedazos de tamaño de bloque. Usamos el método *encipherBlock()* para encriptar desde el buffer de texto plano al buffer del texto cifrado.

Si el texto plano no es múltiplo del tamaño del bloque , debemos encriptar la ultima parte parcial separadamente. Creamos un buffer pequeño que es el tamaño de un bloque , copiamos el dato restante en este y llenamos el resto del buffer con el byte 0x55 (a pesar de esto no hay nada mágico sobre este valor) . Podemos luego usar el método *encipherBlock()* para encriptar el ultimo bloque.

Notar que esta implementación es justo una opción para encriptar un numero grande de datos, y puede ser sobrescrito por una subclase. Esta implementación permite retornar un arreglo de texto encriptado que es múltiplo del tamaño del bloque de encriptación. Hay algoritmos (*ciphertext stealing*) que nos permite producir un resultado que es del mismo tamaño que el de entrada. Este puede ser mas apropiado para algunas situaciones donde nosotros no queremos el *padding* que la implementacion anterior presenta.

El método *encipher* encripta el arreglo de byte plain completo usando el método *encipher().*

El método *decipher* desencripta un bloque de len bytes de datos encriptados cipher comenzando desde el *índice off*, retornando el resultado en texto plano . La longitud del dato encriptado debe ser un múltiplo del tamaño del bloque; este corresponde a la implementación del método *encipher()* que llevael texto cifrado a ser múltiplo del tamaño del bloque. Obviamente, si proveemos un método *encipher()* alternativo debemos también sobrescribir este método.

Como el texto cifrado es un múltiplo del tamaño del bloque , podemos simplemente crear un buffer de texto plano *plain* y desencriptar el cifrado en bloques usando el método *decipherBlock().* Este método no puede determinar si el método *encipher()* completa el dato, el llamador debe habilitar para determinar la correcta longitud del texto plano.

El método *decipher* desencripta el arreglo de byte completo cipher usando el método *decipher().*

El método *blockSize()* debe ser implementado por todo algoritmo de encriptación, e indicar el tamaño del bloque, en bytes ,operado por el algoritmo.

El método *encipherBlock()* encripta un bloque de dato desde *offset po* de arreglo de byte plain en un arreglo de byte cipher de *offset co*. La implementación de este método debe ser capaz de encriptar el dato en el lugar por ejemplo *plain* y *cipher* pueden ser el mismo arreglo.

El método *decipherBlock()* desencripta un bloque de dato de *offset co* de arreglo de bytes cipher en el arreglo de bytes de texto plano de *offset po*.

Esta clase nos provee un template génerico para varios algoritmos de encriptación

7.8.3.- Class DES

La clase DES es una implementación de DES (data encryption standard). Hay varias transformaciones y permutaciones comprometidas claramente requeridas para implementar DES.

DES y DEA son esencialmente la misma cosa ,data encryption standard o data encryption algorithm.

La clase DES es una implementación de DES de alto nivel: esta hace uso de métodos de la clase DEA que actualmente realiza las transformaciones DES de bajo nivel. Esta clase hereda de la clase Cipher e implementa los métodos de encriptacion y desencriptacion de bloque *encipherBlock()* y *decipherBlock()*. Estos métodos serán llamados por los métodos *encipher ()* y *decipher()* de la clase *Cipher*.

Como este algoritmo de encriptación extiende de la clase *Cipher*, podemos usar instancias de esta clase con las clases *CipherOutputStream* y *CipherInputStream* para proveer un canal de comunicaciones DES-encriptado.

```
public class DES extends Cipher {
// public DES ( long key) ...
// public void encipherBlock ( byte[] plain, int po, byte[] cipher, int co) ...
// public void decipherBlock ( byte[] cipher, int co, byte[] plain, int po) ...
// public int blockSize ( ) ...
```

Extendemos la clase *Cipher* , y heredamos las implementaciones por default de los métodos *encipher()* y *decipher()*. Estos métodos hacen uso de los métodos *encipherBlock()* y *decipherBlock()* para encriptar y desencriptar datos en cantidades de tamaño de bloque de encriptación, y solo necesitamos implementar métodos que encriptan y desencriptan un simple bloque de datos a la vez.

DES usa una clave de 56 bits para generar una *key schedule* que es usada en el proceso de encriptación. Esta *key schedule* es una arreglo de 16 entradas de valores de 48 bits que son derivados de la clave inicial.

El constructor acepta la clave de encriptación de 56 bits, y computa y almacena la clave derivada en el arreglo de claves. La clave es actualmente especificada como un valor *long* de 64 bits. Todo octavo bit es un bit de paridad que podemos descartar. Los bits de paridad son usados en cualquier otro lugar para asegurar que una clave ha sido comunicada correctamente.

Debemos usar la misma clave para desencriptar el dato como para encriptarlo, la clave debe ser un secreto que es almacenado solo el por enviador y el receptor.

DES encripta datos en bloques de 64 bits. Este es mas eficientemente implementado usando valores long, mas que arreglos de bytes , este es implementado mas eficientemente de hecho usando los valores long , en vez de arreglos de bytes .Por

lo tanto usamos el método *bytesToLong* de la clase *Crypt* para convertir el texto plano subarray de bytes en un simple valor long. Luego podemos usar el método *encrypt()* , el cual encripta un valor long. Y finalmente usamos el método *longToBytes()* para expandir el resultado encriptado en series de 8 bytes en el arreglo destino cipher especificado.

Descifrar un bloque de dato se ejecuta de la misma forma que encriptar un bloque. Primero concatenamos los 8 bytes del texto cifrado en un valor simple long. Pasamos este a través del método *decrypt(),* el cual retorna el correspondiente texto plano. Podemos luego expandir este valor en el arreglo de salida de texto plano.

DES opera en bloques de 64 bits, por eso debemos retornar el correspondiente tamaño de bloque de 8 bytes. Este es usado para los métodos *encipher()* y *decipher()* para determinar cuantos datos procesa por bloque usando los métodos *decipherBlock()* y *encipherBlock().*

El método *encrypt* realiza la encriptación actual DES. Para encriptar un bloque de dato w debemos primero pasar w a través de una permutación inicial; esto produce un valor permutado x. Luego divide x en 2 mitades; y ejecuta este a través de 16 iteraciones de la función de encriptación DES. Esta función envuelve una expansión, una sustitución S-box , y una permutación P-box; cada iteración usa una clave diferente desde la *key schedule* (clave catálogo) . Después de las 16 iteraciones, juntamos las dos mitades y ejecutamos el pasamos a través de una permutación final para producir el resultado encriptado.

Hacemos este método *public* porque es muy útil para otros códigos .

Por razones de eficiencia, declaramos este método como final y usamos una copia local de la *key schedule* . El compilador puede ejecutar mas optimizaciones con estas declaraciones.

La desencriptación DES envuelve exactamente el mismo proceso, excepto que la *key schedule* es usada a la inversa. El método *decrypt()* es por lo tanto idéntico al método *encrypt(),* excepto que volvemos hacia atrás a través de las claves.

7.8.4.- Class DEA

Esta clase provee funciones *bit-twiddling* de bajo nivel que son usados por la clase DES.

```
public final class DEA {
// public static final long[] makeKeys ( long key) ...
// public static final long initialPerm ( long x) ...
// public static final int desFunc (int x, long k) ...
// public static final long finalPerm ( long x) ...

// protected static final long perm ( long k, int p[] ) ...
// protected static final long rotate ( int l, int r, int s ) ...

// protected static final int keyReducePerm [ ] ...
// protected static final int keyCompressPerm [ ] ...
// protected static final int keyRot [ ] ...
// protected static final int initPerm [ ] ...
// protected static final int finPerm [ ] ...
```

```
// protected static final int sBoxP [ ][ ] ...
```

La característica de esta clase es bastante directa; todos los métodos son estáticos y corresponde a varias etapas del algoritmo de encriptación.

El método *makeKeys()* genera una *key schedule* para la clave especificada key.

El método *initPerm()* realiza la permutación inicial, y el método *finalPerm()* la permutación final. El método *descFunc()* realiza una iteración de la función de encriptación principal. Estas implementaciones usan una tabla especial que combina la permutación de expansión, sustitución S-Box, y la permutación P-Box. Este es significativamente mas rápido que ejecutando los tres pasos separadamente, el cual es como el algoritmo esta definido.

Esta clase hace uso de la función *perm()* que realiza una operación de permutación generalizada, y un método *rotate()* que retorna una valor long que consiste de *l* y *r* cada *s* bits rotados a izquierda y luego se forman juntos en un simple valor de 64 bits.

La clase DEA incluye varios arreglos de números mágicos; estos son usados por el algoritmo de encriptación para realizar permutaciones y substituciones..

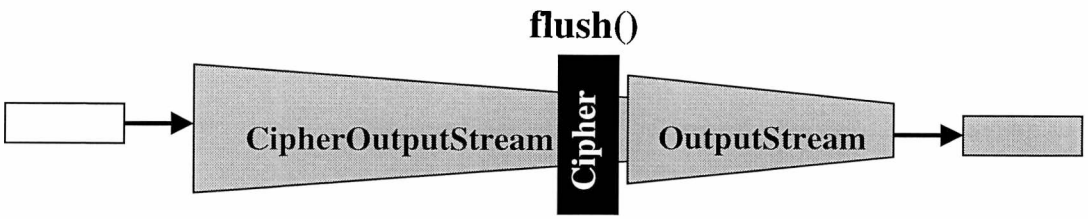
Durante el proceso de generación del *key schedule*, usamos la permutación de reducción de clave *keyReducePerm* que reduce la clave de 64 bits a 56 bits quitando los bits de paridad , la tabla de rotación de la clave keyRot que especifica la cantidad por la cual cada mitad de la clave es rotada a cada iteración y la permutación de la compresión *keyCompressPerm* que selecciona 48 de los 56 bits restantes.

Para la encriptación actual, usamos una permutación inicial initPerm que es una permutación simple de 64 bits de datos, una permutación final finPerm que es la inversa de la permutación inicial, y el arreglo sBoxP el cual combina permutación de expansión DES, S-Box, y P-Box en una tabla simple.

En una implementación simple de DES, la permutación de expansión expande la mitad derecha del dato desde 32 a 48 bits, el S-Boxes reemplaza cada sub-bloque 6 bits con 4 bits de la tabla observada, y p-Box realiza una permutación simple sobre el resultante de 32 bits. Los S-boxes proveen DES con su mayor fuerza; implementa una función no lineal y son diseñados para proteger contra cualquier ataque criptoanalítico. Nosotros apresuramos esta clase combinando estas partes del algoritmo en una tabla simple que lleva a cabo exactamente la misma función. Por supuesto, existen muchas mas posibilidades para la optimización.

7.8.5.- Class CipherOutputStream

Esta clase es un *OutputStream* filtrado que se adjunta a un *OutputStream* existente y encripta todos los datos que se le escriben a él. La clase *Cipher* que es usada para llevar a cabo la encriptación es especificada en el constructor. Esta clase opera almacenando en un buffer todos los datos que son escritos a él, y sólo encriptando y enviando estos datos cuando el método *flush()* es llamado.



Class CipherOutputStream

Esta es similar a la operación de la clase *MessageOutputStream*, excepto que en vez de confiar en un llamado explícito *a send()* enviamos el dato en cualquier momento que el stream sea pasado (con el método *flush()*). El uso del mecanismo *flush* significa que podemos encriptar un canal de comunicaciones en una forma que es completamente transparente al llamador. El correspondiente *CipherInputStream* provee una interface transparente similar al proceso de desencriptación

```
import java.io.*;

public class CipherOutputStream extends FilterOutputStream
{
   // public CipherOutputStream (OutputStream o, Cipher c)...
   // public void flush ( ) throws IOException ...
   // public void close ( ) throws IOException ...
   // protected void writeEncrypted ( ) throws IOException ...
```

La clase *CipherOutputStream* es un stream de salida filtrado; el constructor acepta el *OutputStream* al cual adjuntarlo y la clase *Cipher* con el cual encriptarlo. Describimos los metodos *flush()* y *close()* apropiadamente

En el constructor aceptamos un *OutputStream o* sobre el cual enviaremos los datos encriptados y un *Cipher c* con el cual realizaremos la encriptación. Llamamos al constructor de la superclase con un nuevo *ByteArrayOutputStream byte0* que nos permite almacenar fácilmente todos los datos que son escritos para nosotros, quedando pendiente la encriptación y el envío.

Cuando el método *flush()* es llamado , usamos el método *writeEncrypted( )* para encriptar y escribir el datos que hemos almacenado, y luego pasamos *o (flush)* para asegurar que los datos que hemos escrito son enviados a tiempo.

Nosotros también debemos sobreescribir el método *close()* para pasar (flush) y cerrar el stream adjuntado.

Hay que recordar que hemos adjuntado la superclase *FilterOutputStream* al *byte0*, y entonces la implementación por default de *close()* atentará a cerrar *byte0*; la operación correcta de este método es cerrar *o*.

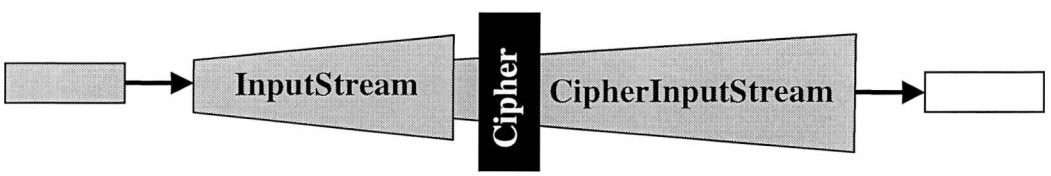
El método *writeEncrypted ( )* encripta y envía los datos que hemos almacenados en el buffer. Si no tenemos datos almacenados entonces simplemente no hacemos nada. De otra manera extraemos los datos almacenados en el buffer, los encriptamos y los enviamos. El copiado extra en este método sirve solo para insertar la longitud del buffer al principio del buffer antes de encriptarlo.

Podemos en realidad encriptar el valor y enviarlo como parte de un mensaje encriptado. Esta divulga menos información. Sin embargo, hay que estar atentos a que el análisis del tránsito pueda en realidad revelar la mayoría de esta información a cualquier que quiera escuchar *'eavesdropper'*, sin importar si encriptamos el mensaje o no.

Llamamos al método *toByteArray()* de *byteO* para extraer los contenidos del buffer dentro de *b*. Luego creamos un nuevo buffer *plain* que contendrá tanto la longitud del buffer como también los contenidos. Escribimos la longitud del buffer en este conjunto utilizando el método. Finalmente podemos encriptar este buffer de texto plano completo el cual incluye la longitud del buffer y escribe el resultado al canal adjunto. El recibidor puede luego desencriptar estos datos y fácilmente determinar cuánta conección ha sido agregada.

7.8.6.- Class CipherInputStream

Esta clase *CipherInputStream* es la clase correspondiente que desencripta los datos del stream. Esta se adjunta a un *InputStream* y desencripta los datos desde éste; los datos deben estar en el formato que está escrito por *CipherOutputStream*. El *Cipher* que realiza la desencriptación se halla especificado en el constructor.



Class CipherInputStream

Esta clase realiza transparentemente la desencriptación; no hay necesidad de llamar explícitamente a cualquier método *receive( )* o su equivalente. En cualquier momento en que *read( )* sea llamado , la clase chequea para ver si se debe desencriptar más información. Si es así, lee y desencripta un nuevo conjunto de datos desde el canal adjunto.

```
import java.io.*;
```

```
public class CipherInputStream extends InputStream {
   // public CipherInputStream (InputStream i, Cipher c)...
   // public int read ( ) throws IOException ...
   // public int read ( byte[] b, int o, int l ) throws IOException ...
   // public int available ( ) throws IOException ...
   // public long skip ( long n ) throws IOException ...
   // public void close ( ) throws IOException ...
   // protected void readEncrypted ( ) throws IOException ...
   // protected ByteArrayInputStream byteI;
   // protected DataInputStream dataI;
   // protected Cipher c;
   // protected void readEncrypted ( ) throws IOException ...
```

Esta clase es una *InputStream* filtrada; no creamos la subclase *FilterInputStream* directamente debido a que ninguna de las implemetaciones del método por default son útiles. En vez de ésto debemos implementar todos los métodos de *InputStream* nosotros mismos.

El *InputStream i* del cual debemos desencriptar los datos , y el *Cipher c* que realiza esta desencriptación , están ambos especificados en el constructor *CipherInputStream.* Adjuntamos una *DataInputStream* a *i* para que podamos hacer uso del método *readFully().*

Cuando debemos desencriptar algunos datos nuevos, los leemos desde *DataInputStream* y los desencripta usando el *Cipher c.* Todos los métodos relevantes en esta clase chequean primero, antes de leer cualquier dato del *byteI* , ver si el buffer se halla en realidad vacío.

Si es así, entonces debemos desencriptar algunos datos más y reemplazar *byteI* con un *ByteArrayInputStream* que lee desde el dato nuevo. Debido a que inicialmente establecemos *byteI* para leer desde cualquier conjunto vacío, el primer llamado para *read()* causará automáticamente un bloque de datos a ser desencriptados.

El método *read()* lee un simple byte desde *byteI* . Primero chequemos si el *byteI* está vacío; si es así debemos desencriptar algunos datos más, y luego llamamos al método *readEncrypted().*

El método *read(b, o, l)* lee un subarreglo de bytes desde *byteI* . Luego chequemos para ver si primero desencriptamos má datos.

El método *available( )* retorna el número de bytes corrientemente disponibles para leer fuera del bloque. Retornamos la cantidad de datos disponibles en *byteI* , sin leer y desencriptar ninguno más. No podemos determinar cuántos datos válidos están disponibles para leer desde un stream adjuntado.

El método *skip(n)* realiza el salto sobre el número de bytes especificado. Debemos primero chequear si el buffer está vacío; si es así, desencriptamos algunos datos más

El método *close()* cierra el stream *dataI* adjuntado

El método *readEncrypted()* lee pedazos de datos encriptados desde *dataI* , desencripta éste, y lo hace disponible para leer desde *byteI.* Este es significativamente complicado por el hecho que actualmente encripta el tamaño de cada pedazo, y debemos desencriptar un bloque de datos antes que podamos determinar cuánto más podría leer y desencriptar.

Para determinar el tamaño de un pedazo, debemos primero desencriptar cuatro bytes de datos. Por lo tanto, creamos un buffer *b* que puede guardar al menos cuatro bytes; el cálculo que usamos crea un arreglo que es de al menos cuatro bytes de longitud, y también es múltiplo del tamaño del bloque de encriptación.

Este método es complicado porque nuestra función *Cipher c* podría igualmente operar con un tamaño de bloque de uno, tres, ocho, u octavos bytes, y debemos estar preparados para cualquier posibilidad.

7.8.7.- Class Cifrado

Esta clase esta compuesta por los métodos encriptar y desencriptar que utilizan los métodos mencionados anteriormente.

El método encriptar toma como parametros de entrada una frase a encriptar y una clave y devuelve la frase encriptada con la clave.

```
public String encriptar( String frase, String cla_entrada) throws IOException {
String palabra=frase;
String clave=cla_entrada;
// convierte de String a Long//
Long claLong= new Long(clave);
// Transforma de Long a long//
long cla=claLong.longValue(); //porque a Des entra un long entonces pasamos de
                                String a long//
ByteArrayOutputStream byte0 = new ByteArrayOutputStream();
DES des= new DES(cla);
CipherOutputStream cipher0 = new CipherOutputStream(byte0,des);
DataOutputStream data0 = new DataOutputStream(cipher0);
data0.writeUTF (palabra);
data0.flush();
return byte0.toString();
}
```

Creamos una variable *byte0* que es de tipo *ByteArrayOutputStream*, que es donde quedará el texto encriptado, luego creamos una variable *cipher0* de tipo *CipherOutputStream* que usa una interface de la clase DES como un algoritmo de encriptación. Cualquier dato que querramos encriptar deberá llamar a *CipherOutputStream.*

El método desencriptar toma como parámetros de entrada una frase encriptada y una clave y devuelve la frase original sin encriptar.

```
public String desencriptar(String frase, String cla_entrada) throws IOException {
byte[] message=frase.getBytes();
String finalmente;
String clave=cla_entrada;
// convierte de String a Long
Long claLong= new Long(clave);
// Transforma de Long a long
```

```
long cla=claLong.longValue(); //porque a Des entra un long entonces pasamos de
                              String a long
DES des= new DES(cla); //genera la clave de DES con cla
ByteArrayInputStream byteI= new ByteArrayInputStream(message);
CipherInputStream cipherI = new CipherInputStream(byteI,des);
DataInputStream dataI= new DataInputStream (cipherI);
finalmente=dataI.readUTF();
return finalmente;
}
```

El método anterior toma una frase encriptada la convierte a *byte[]* y crea un *ByteArrayInputStream* para tomar el dato, luego llama al *CipherInputStream* para desencriptar el dato con la clave generada por DES tomando como base la clave de entrada, una vez que el dato es desencriptado puede leerse desde el *DataInputStream.*

Los dos métodos descriptos anteriormente hacen uso de todos los métodos mencionados en este capítulo y son los que serán invocados por los métodos del paquete *Kerberos* y *Usuarios* para encriptar y desencriptar lo datos.[7]

Conclusiones generales

Consideraciones respecto del trabajo realizado

- Como Internet cobra gran dimensión día a día y permite el paso de información entre personas de diferentes lugares, se le da cada vez mayor importancia a la solución de los problemas de seguridad para garantizar la *privacidad* e *integridad* de la información, por esto nos parece importante el hecho de haber investigado y realizado el desarrollo de los temas anteriormente descriptos.
- Nuestra motivación para desarrollar esta tesis es prevenir el ataque de los hackers a los distintos servicios disponibles en una red, en este momento esto es necesario en función al número de ataques creciente de hackers a Internet, por lo cual surge la utilización de la tecnología actual para este fin.
- Enfocamos específicamente las ventajas que puede aportar el tema de autenticación y autorización Kerberos a los ambientes computacionales de red y lo importante que es realizar una aplicación basada en un protocolo de seguridad. Nosotras implementamos la versión 4 de Kerberos, dejando asentado que la versión 5 aún está en su comienzo y que su vida útil permanece cuestionada.
- El lenguaje que utilizamos para la implementación fue Java el cual tuvimos que investigarlo para realizar esta tesis. A pesar que fue una dificultad debido a que nos llevó bastante tiempo aprenderlo, sabemos que es de gran utilidad interiorizarnos en el mismo y conocer uno de los lenguajes que en los últimos tiempos ha cobrado mayor auge en el mercado, debido a que en Internet se incorporan sistemas interactivos implementados en Java.
- Java permite conectarse a diferentes bases de datos, en un principio utilizamos la base de datos Oracle 7 y aprendimos a realizar los accesos a través del driver correspondiente a dicha base. Además realizamos pruebas en la facultad con la base SQL Server ya que no estaba instalada la base Oracle 7, demostrando así que Java permite conectarse a diferentes bases de datos utilizando el driver correspondiente.
- Java maneja la parte de comunicaciones a nivel de sockets, los que si bien son suficientes para la comunicación 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 esto decidimos usar el paquete de Java RMI (Remote Method Invocation) que abstrae la interface de comunicación a un nivel de llamadas a procedimientos. 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 las mismas.
- Una ventaja de este protocolo es que con un mismo ticket TGS la aplicación puede pedir varios servicios distintos, siempre y cuando el tiempo del TGS no expire, así evita que el usuario reingrese la clave previniendo ataques.

- Los tickets encriptados de Kerberos y la tecnología de pasaje de clave ha alejado a muchos hackers de la violación de la seguridad de algunos de los datos mas sensibles del mundo.
- Debido a que la capacidad y la tecnología de las computadoras que se usan para los ataques avanza día a día, la efectividad y la performance de Kerberos se debe reconsiderar y mejorar continuamente.

Posibles extensiones

Desde el punto de vista de la implementación del protocolo, esta se podría optimizar en varios aspectos:

- Tomamos en forma fija la Master Key del Servidor Kerberos, dicha clave debería cambiarse en determinados lapsos de tiempo.
- Tomamos fija la Identificación del TGS y su clave, como dijimos en el capítulo de Kerberos, este protocolo nos permite trabajar con reinos donde tendríamos varios TGS identificados.
- Usamos el método de encriptación DES para encriptar y desencriptar. La versión 5 de Kerberos permite utilizar distintos métodos de encriptación.
- Para que las aplicaciones fuesen más distribuidas se podrían implementar varios servidores Kerberos cada uno con su expendidor de tickets TGS y sus usuarios autenticados y de esta manera un usuario de un reino podría usar un servicio de otro reino. El esquema requiere que el Server Kerberos en un dominio confíe en el Server Kerberos en el otro dominio para autenticar sus usuarios, también los servers participantes en el segundo dominio deberán confiar en el server Kerberos del primer dominio.
- Se puede ampliar esta implementación para lograr implementar la V5 de Kerberos.
- Para probar nuestra implementación del protocolo Kerberos hicimos una aplicación muy sencilla, que lo único que hace es permitir ingresar los parámetros necesarios para que Kerberos pueda otorgar seguridad y muestra una clave de acceso o un mensaje de error, se puede mejorar esta implementación haciendo una aplicación kerberizada que tome la clave de acceso al servicio y permita usar el servicio por el tiempo estipulado en el ticket. Por lo tanto las aplicaciones kerberizadas son las que controlarán que los tiempos de los tickets no terminen, y en caso que terminen cortarán el uso del servicio.

Problemas que se nos presentaron

- La investigación del lenguaje Java nos llevó mucho tiempo ya que nunca lo habíamos utilizado. En el transcurso de dicho aprendizaje tuvimos que interiorizarnos en diversos temas que nos generaron algunas dificultades:
  - Entender la metodología de RMI, para los accesos a métodos remotos.
  - Acceder a la base de datos con los drivers JDBC/ODBC a través de la herramienta Visual Age for Java utilizada, ya que no ve la variable de entorno CLASSPATH donde especificamos el path de los drivers (según los papers encontrados), sino que debimos importar los drivers al ambiente como paquetes, lo cual nos llevo bastante tiempo descubrirlo.
- El problema del tiempo de vida del ticket es la manera de elegir entre seguridad y conveniencia, si el tiempo de vida del ticket es largo entonces si el ticket y su clave de sesión asociados son robados o extraviados ellos pueden ser usados durante un largo período de tiempo. Tal información puede ser robada si el usuario se olvida de salir de la workstation. Alternativamente si el usuario ha sido autenticado sobre un sistema que permite usuarios múltiples, otro usuario con acceso al root puede estar disponible para encontrar la información necesaria para usar tickets robados. El problema con un ticket con un corto tiempo de vida es que cuando este expira, el usuario tendría que obtener uno nuevo y para hacerlo debería entrar su password.
- Los temas de seguridad no fueron introducidos en la carrera y en particular, la seguridad en redes implementada por medio de una arquitectura distribuida, resultaron de una mayor dificultad de lo que se impuso inicialmente.
- Entender y diseñar la arquitectura distribuida que utiliza Kerberos.

Bibliografía

[1] Network and Internetwork Security Principles and Practice
Williams Stallings, Ph.D.
Los temas investigados en este libro son Criptografía, en especial el algoritmo de encriptación DES y el Protocolo de Autenticación y Autorización Kerberos (Versión 4 y Versión 5).

[2] Redes Globales de Información con Internet y TCP/IP
Douglas E. Comer
Tercera Edición
En este libro investigamos el tema de seguridad en Internet.

[3] Computer Networks
Andrew S. Tanenbaum
Third edition
Describe el tema de criptografía, en especial el algoritmo DES, RSA e IDEA. Hace referencia también a la autenticación utilizando el protocolo de Kerberos.

[4] Core JAVA (1.1) Volumen 1 – Fundamentals
Cay S. Horstmann – Gary Cornell
1997 Sun Microsystem, Inc
En este libro hay varios ejemplos de java que nos ayudaron a entender el lenguaje. También encontramos una buena explicación sobre el tema de RMI (Acceso a Métodos Remotos con Java) y JDBC (Acceso a las Bases de Datos mediante JDBC/ODBC)

[5] Manual de JAVA
Patrick Naughton
1996 McGraw-Hill Interamericana de España S.A.
Con este manual nos introducimos al lenguaje de programación Java.

[6] Internet y Seguridad en Redes
Karajit Siyan, Ph.D. – Chis Hare
Encontramos los principios básicos para implementar una política de seguridad en red.

[7] JAVA Network Programming
Merlin, Conrad Hughes, Michael Shoffner, María Winslow
Describe los fundamentos necesarios para implementar en Java el algoritmo de encriptación DES.

[8] Paper : Kerberos User's Frequently Asked Questions
Septiembre, 1995

Este Paper tiene varios items que explican el funcionamiento del Protocolo Kerberos.

[9] http://www.ov.com/misc/krb-faq.html
Paper : An Authentication Service for Computer Networks
B. Clifford Neuman and Theodore Ts'o
Septiembre, 1994
Este paper explica el Modelo de Autenticación y Autorización Kerberos, en su versión 4.

[10] http://nii.isi.edu/publications/kerberos-neuman-tso.html
Paper : Kerberos Authentications and Authorization System
by Miller, Neuman, Schiller and Saltzer
1985 – 1987
Este paper explica la versión 4 del Modelo de Autenticación y Autorización Kerberos y su forma de autenticar mediante tickets y claves de sesión.

[11] Paper : An Authentication Service for Open Network System
Jennifer G. Steiner
Este paper nos brindó información sobre servicios de Autenticación mediante claves y pasaje de tickets

[12] http://www.ibm.com/java/education/
En esta página encontramos información sobre JDBC para invocación de métodos remotos en Java

[13] http://www.fie.us.es/info/internet/JAVA/
En este paper encontramos información sobre el lenguaje Java, manejo de clases e interfaces

[14] http://www.redbooks.ibm.com/
En este paper encontramos información acerca del uso del Driver JDBC, y de la implementación y uso de RMI (Acceso a Métodos Remotos)

[15] http://www.java.sun.com/products/jdk/
En este paper tiene información sobre JDK y JDBC.

[16] http://www.java.sun.com/products/jdk/1.1/docs/guide/rmi/spec/rmi-intro.doc.html
En este Paper esta desarrollado el tema de RMI (Invocación a Métodos Remotos)

[17] http://www-supdev.us.oracle.com/
En este paper encontramos como trabajar para acceder a las Bases de Datos Oracle mediante JDBC

[18] http://www.software.ibm.com/data/db2/java/index.html
En este paper está desarrollado el tema de Java Database Connectivity (JDBC) con varios ejemplos que fueron de gran utilidad.

[19] http://www.oracle.com.sg/nca/java_nca/
En este paper está desarrollado el tema de Acceso Relacional de Datos desde Java con JDBC y SQLJ

[20] http://www.redbooks.ibm.com/SG245081/javaec04.fm.html
En este paper esta muy bien explicado el tema de JDBC, en forma teórica y práctica.

[21] http://www.inst.bekerley.edu/usr/pub/kerberos
En este paper esta desarrollado el tema de la versión 4 de Kerberos y sus diferencias con la versión 5.

[22] http://java.sun.com/docs/books/tutorial/ui/overvieu/conpHierarchi.html
Este paper desarrolla el tema de DES (Data Encriptación Standard)

[23] Paper, Kerberos *ICSA - 750 Earl Ipsaro*
Abril 1996
En este paper encontramos información relacionada con el protocolo de Autenticación y Autorización Kerberos.